Antonia Albani
Jan L.G. Dietz
Jan Verelst (Eds.)

LNBIP 79

Advances in Enterprise Engineering V

First Enterprise Engineering Working Conference, EEWC 2011
Antwerp, Belgium, May 2011
Proceedings

Springer

Lecture Notes
in Business Information Processing 79

Antonia Albani
Jan L.G. Dietz
Jan Verelst (Eds.)

Advances in Enterprise Engineering V

First Enterprise Engineering
Working Conference, EEWC 2011
Antwerp, Belgium, May 16-17, 2011
Proceedings

Springer

Volume Editors

Antonia Albani
University of St. Gallen
9000 St. Gallen, Switzerland
E-mail: antonia.albani@unisg.ch

Jan L.G. Dietz
Delft University of Technology
2628 CD Delft, The Netherlands
E-mail: j.l.g.dietz@tudelft.nl

Jan Verelst
University of Antwerp
2000 Antwerp, Belgium
E-mail: jan.verelst@ua.ac.be

ISSN 1865-1348 e-ISSN 1865-1356
ISBN 978-3-642-21057-0 e-ISBN 978-3-642-21058-7
DOI 10.1007/978-3-642-21058-7
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011926784

ACM Computing Classification (1998): J.1, H.3.5, H.4

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Enterprise engineering is an emerging discipline that studies enterprises from an engineering perspective. It means that enterprises are considered to be designed and implemented systems, which consequently can be re-designed and re-implemented if there is a need for change. Enterprise engineering is rooted in both the organizational sciences and the information system sciences. The rigorous integration of these traditionally disjoint scientific areas has become possible after the recognition that communication is a form of action. The operating principle of organizations is that actors enter into and comply with commitments, and in doing so bring about the business services of the enterprise. This important insight clarifies that enterprises belong to the category of social systems, i.e., its active elements (actors) are social individuals (human beings). The unifying role of human beings makes it possible to address problems in a holistic way, to achieve unity and integration in bringing about any organizational change.

Also when regarding the implementation of organizations by means of modern information and communication technology (ICT), enterprise engineering offers innovative ideas. In a similar way as the ontological model of an organization is based on atomic elements (namely, communicative acts), there is an ontological model for ICT applications. Such a model is based on a small set of atomic elements, such as data elements and action elements. By constructing software in this way, the combinatorial effects (i.e., the increasing effort it takes in the course of time to bring about a particular change) in software engineering can be avoided.

The development of enterprise engineering requires the active involvement of a variety of research institutes and a tight collaboration between them. This is achieved by a continuously expanding network of universities and other institutes, called the CIAO! Network (www.ciaonetwork.org). Since 2005 this network has organized the annual CIAO! Workshop, and since 2008 its proceedings have been published as *Advances in Enterprise Engineering* in the Springer LNBIP series. From 2011 on, this workshop was replaced by the Enterprise Engineering Working Conference (EEWC). This book contains the proceedings of the first EEWC, which was held in Antwerp, Belgium.

May 2011

Antonia Albani
Jan L.G. Dietz
Jan Verelst

Enterprise Engineering – The Manifesto

Introduction

This manifesto presents the focal topics and objectives of the emerging discipline of enterprise engineering, as it is currently theorized and developed within the CIAO! Network. There is a close cooperation between the CIAO! Network (www.ciaonetwork.org) and the Enterprise Engineering Institute (www.ee-institute.com) for promoting the practical application of enterprise engineering. The manifesto comprises seven postulates, which collectively constitute the *enterprise engineering paradigm* (EEP).

Motivation

The vast majority of strategic initiatives fail, meaning that enterprises are unable to gain success from their strategy. Abundant research indicates that the key reason for strategic failures is the lack of coherence and consistency among the various components of an enterprise. At the same time, the need to operate as a unified and integrated whole is becoming increasingly important. These challenges are dominantly addressed from a functional or managerial perspective, as advocated by management and organization science. Such knowledge is necessary and sufficient for managing an enterprise, but it is inadequate for bringing about changes. To do that, one needs to take a constructional or engineering perspective. Both organizations and software systems are complex and prone to entropy. This means that in the course of time, the costs of bringing about similar changes increase in a way that is known as combinatorial explosion. Regarding (automated) information systems, this has been demonstrated; regarding organizations, it is still a conjecture. Entropy can be reduced and managed effectively through modular design based on atomic elements. The people in an enterprise are collectively responsible for the operation (including management) of the enterprise. In addition, they are collectively responsible for the evolution of the enterprise (adapting to needs for change). These responsibilities can only be borne if one has appropriate knowledge of the enterprise.

Mission

Addressing the challenges mentioned above requires a paradigm shift. It is the mission of the discipline of enterprise engineering to develop new, appropriate theories, models, methods and other artifacts for the analysis, design, implementation, and governance of enterprises by combining (relevant parts of) management and organization science, information systems science, and computer science. The ambition is to address (all) traditional topics in said disciplines

from the enterprise engineering paradigm. The result of our efforts should be theoretically rigorous and practically relevant.

Postulates

Postulate 1

In order to perform optimally and to implement changes successfully, enterprises must operate as a unified and integrated whole. *Unity* and *integration* can only be achieved through *deliberate enterprise development* (comprising design, engineering, and implementation) and *governance*.

Postulate 2

Enterprises are essentially social systems, of which the elements are human beings in their role of *social individuals*, bestowed with appropriate *authority* and bearing the corresponding *responsibility*. The *operating principle* of enterprises is that these human beings enter into and comply with *commitments* regarding the products (services) that they create (deliver). Commitments are the results of *coordination acts*, which occur in universal patterns, called *transactions*.

Note. Human beings may be supported by technical artifacts of all kinds, notably by ICT systems. Therefore, enterprises are often referred to as socio-technical systems. However, only human beings are responsible and accountable for what the supporting technical artifacts do.

Postulate 3

There are two distinct perspectives on enterprises (as on all systems): *function* and *construction*. All other perspectives are a subdivision of one of these. Accordingly, there are two distinct kinds of models: *black-box models* and *white-box models*. White-box models are *objective*; they regard the construction of a system. Black-box models are *subjective*; they regard a function of a system. *Function is not a system property* but a relationship between the system and some stakeholder(s). Both perspectives are needed for developing enterprises.

Note. For convenience sake, we talk about the business of an enterprise when taking the function perspective of the customer, and about its *organization* when taking the construction perspective.

Postulate 4

In order to manage the complexity of a system (and to reduce and manage its entropy), one must start the constructional design of the system with its *ontological model*. This is a fully implementation-independent model of the *construction* and the *operation* of the system. Moreover, an ontological model has a *modular* structure and its elements are (ontologically) *atomic*. For enterprises the

meta-model of such models is called *enterprise ontology*. For information systems the meta-model is called *information system ontology*.

Note. At any moment in the life time of a system, there is only one ontological model, capturing its actual construction, though abstracted from its implementation. The ontological model of a system is comprehensive and concise, and extremely stable.

Postulate 5

It is an *ethical necessity* for bestowing authorities on the people in an enterprise, and having them bear the corresponding responsibility, that these people are able to *internalize* the (relevant parts of the) *ontological model* of the enterprise, and to constantly validate the correspondence of the model with the operational reality.

Note. It is a duty of enterprise engineers to provide the means to the people in an enterprise to internalize its ontological model.

Postulate 6

To ensure that an enterprise operates in compliance with its *strategic concerns*, these concerns must be transformed into generic functional and constructional *normative principles*, which guide the (re-)development of the enterprise, in addition to the applicable specific requirements. A coherent, consistent, and hierarchically ordered set of such principles for a particular class of systems is called an *architecture*. The collective architectures of an enterprise are called its *enterprise architecture*.

Note. The term "architecture" is often used (also) for a model that is the outcome of a design process, during which some architecture is applied. We do not recommend this homonymous use of the word.

Postulate 7

For achieving and maintaining unity and integration in the (re-)development and operation of an enterprise, organizational measures are needed, collectively called *governance*. The *organizational competence* to take and apply these measures on a continuous basis is called *enterprise governance*.

May 2011 Jan L.G. Dietz

Organization

EEWC 2011 was the First Working Conference resulting from a series of successful CIAO! Workshops over the years. These workshops were aimed at addressing the challenges that modern and complex enterprises face in a rapidly changing world. The participants of the workshops shared a belief that dealing with these challenges requires rigorous and scientific solutions, focusing on the design and engineering enterprises.

This conviction has led to the idea of organizing an international working conference on the topic of enterprise engineering, in order to bring together all stakeholders interested in making enterprise engineering a reality. This means that not only scientists were invited, but also practitioners. It also means that the conference is aimed at active participation, discussion and exchange of ideas in order to stimulate future cooperation among the participants. This makes EEWC a working conference contributing to the further development of enterprise engineering as a mature discipline.

The organization of EEWC 2011 and the peer review of the contributions made to EEWC 2011 were accomplished by an outstanding international team of experts in the fields of enterprise engineering.

General Chair

Jan L.G. Dietz	Delft University of Technology, The Netherlands

Organization Chairs

Herwig Mannaert	University of Antwerp, Belgium
Jan Verelst	University of Antwerp, Belgium

Program Chair

Antonia Albani	University of St. Gallen, Switzerland

Program Co-chairs

Eduard Babkin	Higher School of Economics Nizhny Novgorod, Russia
Junichi Iijima	Tokyo Institute of Technology, Japan
José Tribolet	INESC and Lisbon University of Technology, Portugal

Program Committee

David Aveiro
Eduard Babkin
Joseph Barjis
Bernhard Bauer
Emmanuel delaHostria
Eric Dubois
Johann Eder
Joaquim Filipe
Rony G. Flatscher
Ulrich Frank
Remigijus Gustas
Birgit Hofreiter
Jan Hoogervorst
Stijn Hoppenbrouwers
Christian Huemer
Junichi Iijima
Peter Loos
Florian Matthes
Graham Mcleod
Aldo de Moor
Hans Mulder
Martin Op 't Land
Pontus Johnson
Erik Proper
Gil Regev
Pnina Soffer
José Tribolet
Jan Verelst
Robert Winter
Marielba Zacarias

Table of Contents

Designing Organizations with DEMO

Combining DEMO with Other Methods

Studies in Enterprise Architecture

Designing the Information Organization from Ontological Perspective

Joop de Jong[1,2]

[1] Delft University of Technology,
P.O. Box 5031, 2600 GA Delft, The Netherlands
[2] Mprise
P.O. Box 598, 3900 AN Veenendaal, The Netherlands
jdjong@mprise.nl

Abstract. Actors act on the organization's world. Information that is needed for performing coordinating actions of those actors is equal to factual knowledge that consists of either facts that constitute the world's states or facts that are derived from these original facts. That means that business actors which are directly responsible for production results are indirectly responsible for the information that is based on their production results. This paper proposes a way of working by which the design of the ontological model of the information organization is based on the ontological model of the business organization. Central to the proposed approach is the concept of responsibility. This is a fundamentally different approach than many current approaches in practice to which information is usually extracted from a database of data and from whom nobody has any idea of the accountability regarding the reliability, timeliness, etc. of the data.

Keywords: enterprise engineering, enterprise ontology, information management, way of working, DEMO.

1 Introduction

This paper focuses on an important theme in the domain of the enterprise engineering, defined in the manifest [1], namely the way of connection between business actors which request for information and business actors which create relevant facts for computing the information. Intellectual actors which perform infological production actions such as remembering, reproducing, reasoning and computing on this information as well as documental actors which perform datalogical production actions such as copying, storing, and retrieving data are actively involved in the process of generating information. The interwork of those types of actors leading to an optimal performance of the enterprise requires a thorough construction design of the enterprise. Such a design can be made by using the methodology that is called the Design and Engineering Methodology for Organizations, DEMO for short. It is developed by Dietz [2, 3]. He focused on the way of thinking and the way of modeling in order to be able to design an ontological model of the business organization of the enterprise. However, the proposed principles are also applicable in

A. Albani, J.L.G. Dietz, and J. Verelst (Eds.): EEWC 2011, LNBIP 79, pp. 1–15, 2011.

designing the ontological model of the information organization of the enterprise. In this paper we present a methodological approach or a way of working to construct the model of the information organization from an ontological perspective. This way of working distinguishes from others in at least the following two points. First, the design of the construction model of the information organization is achieved from the ontological model of the business organization. Second, several types of actors (business, intellectual and documental) work together to create and remember original facts, to derive new facts from existent facts, and to derive information from facts for supporting conversations at the business level. Each of them has their own responsibilities in this collaboration network. In particular, business actors create new facts and are thus liable for the quality of these facts. They have the shared responsibilities and the liabilities for the quality of all information. This approach differs fundamentally from the current approach in practice to which information is usually extracted from a database of data and from whom nobody has any idea about the responsibility regarding the reliability, timeliness, etc. of the data.

As we said we base the proposed way of working on DEMO. DEMO is based on the relatively young research field of Language Action Perspective, or LAP for short [4-8]. The focus on communications the concept for understanding and modeling of organizations comes from the Speech Act Theory [9, 10] and the Theory of Communicative Action [11, 12]. This theory considers speech acts as a vehicle to act. Mulder has compared the most important LAP-based theories [13] and comes to the conclusion that DEMO is the most appropriate methodology to (re)design organizations in an integrated way. DEMO understands an organization as a construct that operates in a social world, also called an inter-subjective world. Within the inter-subjective world human subjects perform actor roles and interact with each other by transactions. This also means that the need for information is considered from the perspective of coordination.

This paper is structured as follows. Firstly, chapter 2 contains a summary of the ψ-theory. DEMO has been grounded on this theory. For that reason it is necessary for a good understanding of the proposed way of working to treat briefly it's most important and relevant concepts. The proposed way of working in order to design the information organization is discussed in section 3. It is split in four steps. Each step is elaborated separately in a different section. Section 4 contains some concluding remarks and directions for further research.

2 Summary of the ψ-Theory

For a good understanding of this paper a summary of a part of the ψ-theory on which Dietz [2] based the DEMO methodology is presented. The ψ-theory consists of four axioms, viz. the operation axiom, the transaction axiom, the composition axiom and the distinction axiom, and the organization theorem.

The *operation axiom* states that the operation of the organization is constituted by the activities of actors, which are elementary chunks of authority and responsibility fulfilled by human beings. Actors perform two kinds of acts: production acts, or P-acts for short, and coordination acts, or C-acts for short. These acts have definite results, namely production facts and coordination facts, respectively. By performing

P-acts, actors contribute to bringing about goods or that are delivered to other actors. By performing C-acts, actors enter into and comply with commitments towards each other regarding the performance of P-acts. The *transaction axiom* states that coordination acts are performed as steps in universal patterns. These patterns, also called transactions, always involve two actor roles. They are aimed at achieving a particular result, the P-fact. One of the two partaking actor roles is called the initiator, the other the executor of the transaction. In the order phase, the initiator and the executor pursue to reach agreement about the P-fact that the executor is going to bring about as well as the intended time of creation. In the execution phase, the executor brings about this P-fact. In the result phase, the initiator and the executor pursue to reach agreement about the P-fact that is actually produced as well as the actual time of creation (both of which may differ from the requested one). Only if this agreement is reached will the P-fact become existent. The *composition axiom* states that every transaction is enclosed in some other transaction, or is a customer transaction of the organization under consideration, or is a self-activation transaction. The *distinction axiom* states that there are three distinct human abilities playing a role in the operation of actors, called performa, informa and forma. The forma ability is the human ability to conduct documental actions, such as storing, retrieving, transmitting, etc. These are actions by which the content of the documents or data is of no importance. Actors which use the forma ability to perform P-acts are called documental actors or D-actors for short. The informa ability is the human ability to conduct intellectual actions, such as reasoning, computing, remembering and recalling of knowledge, etc. These are actions by which the content of data or documents, abstracted from the form aspects, is of importance. Actors which use the informa ability to perform P-acts are called intellectual actors, or I-actors for short. The performa ability is the human ability to conduct new, original actions, such as decisions, judgments etc. The performa ability is considered as the essential human ability for doing business, of any kind. Actors which use the performa ability to perform P-acts are called business actors or B-actors for short. The last part of the ψ-theory that we would like to explain is the *organization theorem*. It states that the organization of an enterprise is a social system that is constituted as the layered integration of three homogeneous systems: the B(usiness)-organization, the I(ntelligent)-organization, and the D(ocument)-organization. The D-organization supports the I-organization, and the I-organization supports the B-organization. All three systems are called aspects systems of the total organization of the enterprise.

A system can be considered from two different perspectives, namely from the function perspective or from the construction perspective. The function perspective on a system is based on the teleological system notion which is concerned with the (external) behavior or performance of a system. This notion is adequate for the purpose of controlling or using a system. It is about services or products which are delivered by the system. The construction perspective on a system is based on the ontological system notion. Systems have to be designed and constructed. During the design and engineering process questions of being effectively and efficiency of the chosen design have to be answered by the constructor of the system. Our point of departure in this paper is the ontological system notion. The integration between the three organizations is established through the cohesive unification of human being [14].

Based on these axioms, DEMO designs the ontological model of any social system by four coherent and consistent models [3]. First, the *Construction Model (CM*) contains the identified actor roles in the composition and in the environment, the identified transaction kinds among the actor roles as well as the information links from the actor roles in the composition to the internal fact banks (i.e. the production banks and coordination banks of the identified transaction kinds), and to the identified external fact banks. The CM is a constitution of two models, namely de *Interaction Model (IAM)* and the *Interstriction Model (ISM).* Second, the *Process Model (PM)* of an organization is the ontological model of the state space and the transition space of the coordination world of the organization. Third, the *Action Model (AM)* of an organization contains action rules for every agendum kind for every identified actor role. An action rule specifies the (production and/or coordination) acts that must be performed and the corresponding conditions (in the production world and/or the coordination world) that must hold, on the occurrence of an instance of the agendum kind (which is an event in the coordination world). Fourth, the *Fact Model (FM)* of an organization is the ontological model of the state space and the transition space of the production world of the organization.

3 The Way of Working

3.1 Introduction

Designing an effective enterprise implicates that business processes and information processes must be defined in coherence. The correlation between these two kinds of processes is described in the AM. The information functions in the AM are related to information processes which input consists of original or derived facts which are retrieved from fact banks inside and outside the organization boundary. These fact banks are filled by business actors which perform production acts. In other words, business actors need (derived) facts as information parts which have been created earlier by themselves or by other business actors. The proposed way of working that leads to the design of the ontological model of the information organization consists of four steps. The four steps are the following:

1. *Determining information functions.* This step results to a complete list of information functions which are requested by the business actors inside the boundary of the organization. According to the Generic System Development Process [2, 15] the list can be considered as the function model of the object system that corresponds with the information organization. It is derived from the construction model of the user system that corresponds with the ontological design of the business organization of the enterprise. The function model contains two different kinds of information functions. Firstly, the functions which reproduce original facts and perform infological actions on these facts to provide information to business actors. Secondly, the functions which remember the original facts created by business actors.

2. *Analyzing information functions.* Each information function from the mentioned list relates to an information process. The information processes which deliver information to business actors are powered by original facts created by business actors. Those business actors are responsible for the data they created. That makes business actors shared responsible for the quality of the available information. For each information process the responsible business actors which contribute to the quality of the output of the information process are determined.
3. *Remembering by business actors.* After determining the contribution of each business actor to each information function all fact kinds per business actor can be constituted. All these fact kinds, added with object classes and result kinds, are described in the Bank Contents Table (BCT) [2]. This table contains the input data for modeling the FM subsequently. The FM relates the fact kinds, object classes and result kinds according to the ontological coexistence rules to each other. It presents a coherent and comprehensive picture of the object world which has been obtained as a result of a methodological way of working. This approach is completely different from what one often see in practice where the principle of the waiter strategy [2] still remains very popular.
4. *Modeling information processes.* After specifying both the information processes and the fact banks in which the original facts that corresponds with specific information processes are remembered and reproduced, the IAM of the I-organization can be designed. This step results in this model which provide clear insights both into the possibilities of reuse infological actions and in the responsibility areas of the individual actors.

Those four steps are elaborated in the next four sections.

3.2 Determining Information Functions

The execution of an action rule depends on the state of the coordination world or the production world on which the organization performs its actions. Querying the relevant state of a world takes place through an information process. According to Dietz [2] the AM is the most detailed and comprehensive aspect model. It is atomic on the ontological level. This leads to our perception that the direct interrelationship between the B-organization and I-organization lies in the AM.

Let us elaborate this topic by an example. Figure 1 exhibits the IAM of a small part of an education center, for short EC, which offers standard courses to employees of a known set of companies (clients). A client fills in the enrollment form and sends it to EC. By sending a confirmation email, it is conceived that EC gives a promise for the corresponding enrollment. Each enrollment should be paid by the client. For some clients a particular discount is applicable. In the payment request, this fact should be taken into account. The result kinds of the transaction kinds are given in Table 1. Figure 2 exhibits the PM. For each process step is asked under what conditions the corresponding C-act may be activated. For example, the actor A01 receives a request and is put to the choice whether or not a promise action to do. The answer is gained by executing a Boolean logical expression in which one or more information functions are called.

Table 1. Transaction Result Table of EC

•	Transaction kind	Result kind
•	T01 Enrollment	R01 Enrollment [E] has been performed
•	T02 Discount approval	R02 Discount for [E] has been approved
•	T10 Payment	R10 Enrollment [E] has been paid

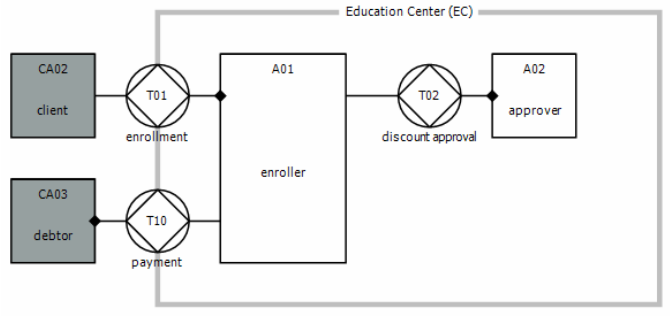


Fig. 1. The Interaction Model of EC

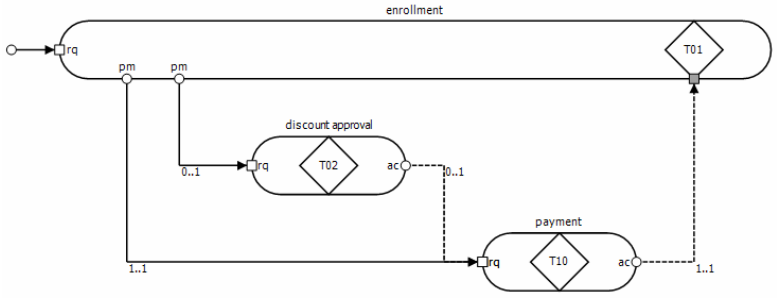


Fig. 2. The Process Model of EC

The concerning Boolean logical expression is part of the action rule which is defined in the AM. The action rule related to the agendum kind (T01, requested) shows that both a client data check and a check whether the desired course is already full booked must be carried out. For checking several pieces of information are required the client data, the list of standard courses offered by EC, a list of companies which are known at EC, the number of current participants and the maximum allowable capacity of the standard course. All those pieces of information designate actually facts which are results of decisions taken by business actors in the past. Transactions could also be done in the mind of people. For example, the client CA02 decided to order a standard course for one of its employees at EC. The fact which reflects this production result is remembered by CA02 itself.

DEMO distinguishes several transaction statuses, such as requested, promised, declined, stated, rejected, accepted, and others, which could be part of an agenda kind. Agenda kinds (transaction kind, transaction status) contain a set of action rules and all

A01	Enroller
	when T01 **of** new(E) **is** requested **if** *the client.course is equal to a valid standard course* ***and*** *the client.company is known at EC* ***and*** *#participants(standard course) < max_capacity(standard course)* **then** T01 **of** [E] **must be** promised **else** T01 **of** [E] **must be** declined

Fig. 3. Example of Action Rule of EC

action rules have their own Boolean logical expression. Based on those expressions a comprehensive and consistent list of required information functions can be constituted. Figure 3 exhibits one of the action rules of agendum (T01, requested).

Table 2 is created from Figure 3. It contains the list of information functions which are called by the action rules in the AM of EC. The information functions 1 through 3 and 5 correspond to an original fact and the fourth information function corresponds to a derived fact. Corresponding to a derived fact means that the output of the information function is computed by of one or more intelligent actors based on one or more original facts. Take into account that only information functions which are needed by internal actors are determined. The information needs of external actors are unknown and by definition out of scope.

Table 2. List of Required Information Functions

Actor	No	Required Information Functions	Short Description
A01	1	Client data	The client data includes the identification of the client, the client company, the participant, and the standard course the client likes to order for the participant
	2	Standard courses of EC	The standard course that the client likes to order must be available at EC
	3	The companies the EC works for	The client company must be known at EC
	4	Number of attendees which are enrolled for Standard course	The total amount of enrollments for the requested standard course
	5	Maximum capacity for standard Course	EC has used for any standard course a maximum number of participants
A02		……..	

Next, we would like to pay attention to the criteria satisfying an information function for being acceptable within a Boolean logical expression. We indicated already that an information process that corresponds with an information function enquires about the state of a part of the production world or of the coordination world. The issue we would like to discuss is about the distinctions between what we call coordination-specific information and competence-specific information. Should we consider both kinds of information as relevant to be included in the way of working we propose? The important question is to what extent the competencies defined for an

actor role affect the definitions of the information functions that are part of the action rules which are linked to this actor role. One would say that information gained by an actor in the inter-subjective world of cooperating actors is still known by the actor as it works on a production act in the subjective world. That is certainly true, but that does not mean that all information you would like to have available during the production act is available in the inter-subjective world. We restrict ourselves to the information which is obtainable in the inter-subjective world. Let us look again at the EC case. The approver determines whether a client receives a particular discount. There is nothing about the approval written on paper. Company guidelines regarding this process are only in heads of human being. In practice the approver takes its own decisions. Two distinct situations can occur in practice. They are exhibit by Figure 4.

A02	**Approver**
	when T02 **of** [E] **is** requested T02 **of** [E] **must be** promised

A02	**Approver**
	when T02 **of** [E] **is** requested **if** *guidelines and documented experiences are available* **then** T02 **of** [E] **must be** promised **else** T02 **of** [E] **must be** declined

Fig. 4. Two Action Rules for distinct Situations

The first action rule shows that the management of EC expects that the approver memorizes all guidelines. The approver must be well experienced by cases from the past. It knows many clients and knows very well which clients always ask for discount and which clients always accept the offer of the EC automatically. The experiences of the actor have a strong contribution in judging the approval request. Behind the second action rule there is another narrative. The management of EC concluded that finding an approver with such a high skills as written in the first situation is very difficult because high competent people ask for high salaries and EC is not able to pay a high salary to the approver. They found a solution by reducing the expectations regarding the quality of the skills of the approver. They would provide the new approver both the guidelines for approval and some other practical guidelines on paper. They are convinced that the combination of lower skilled people together with guidelines on paper provide an equal quality in approval. In case of the second action rule (cf. Fig. 4) the approver seems to compensate a lack of experiences and knowhow by using some tools during its production act to provide the same contribution as the actor of the first action rule. This reasoning sounds correct but is in fact completely

wrong. What actually happens is that one tries to solve a problem that occurs in the subjective world of the actor by an additional information request in the inter-subjective world. This is incorrect.

The narrative behind the second action rule should be resolved within the subjective world. In the subjective world a less qualified person who is supported by standard tools, relevant manuals and guidelines, etc. must be linked to the actor A02. The concerning manuals do not contain information as we understand within the framework of this paper. They contain competence-specific information which is opposed to coordination-specific information that we discuss in this paper.

3.3 Analyzing Information Functions

Each information function is related to an information process within the organization. An information process enquires business actors and gains original facts created by business actors inside and outside the organization boundary as input data. A business actor which create facts is owner of those facts and therefore responsible for the quality of those facts and subsequently also responsible for the quality of the information those facts contribute. Business actors, which request for information, ought to ask permission for using facts which contribute to the concerning information. Only when the facts are correctly applied within the information function, the authorization for using those facts should have been approved. On the other hand, the actors which request for information may expect that the quality of the available facts is sufficient in terms of reliability, accuracy, etc. As an example, the sender of an important package to a customer may assume that the address that he receives from an intellectual actor will be the most recent address of that customer. If the address appears to be incorrect the business actor which is responsible in the organization for updating customer data should therefore be liable. Just another example, the salesman in a company who is able to sell a product may assume that the concerning product is available on stock if the common group of actors which affect the inventory of that product have made clear that at least one qualified product in the magazine would be available. This common group of actors must be considered as liable for the consequences if such a product does not appear available. Intellectual actors which deliver information to business actors do not only transfer original facts to those business actors but they also perform production actions on original facts. These kinds of production actions are also called intellectual actions or infological actions. Intellectual actors operate under the rules of logic and therefore they demonstrate predictable behavior. In practice, the acts of intellectual actors are often taken over by software applications. Of course, intellectual actors also have their own responsibility for the quality of their infological actions, but given the nature of their actions, the quality issue can often be compensated by properly programmed (parts of) actors.

This step in the proposed way of working leads to the acquiring of at least two different views. First, it gives per information function information about the responsible business actors which contribute to the quality of the concerning information function. Second, it gives more information about the interdependencies between business actors within the organization and between business actors inside and outside the organization boundary. Insight into the interdependencies is at least in

two different situations of utmost importance. Firstly, it is important in a situation where it is desirable to change the ontological construction model of the enterprise. A change of a role definition may affect acts of business actors which coordination actions depend on the availability of specific information based on original facts created by actors which role definition has been adapted. An actor role even can be outsourced. This could have major consequences for the availability of information for other actors within the organization. Secondly, in a situation that operational disruption occurs in the execution of an actor role. If there is a delayed delivery of raw materials, or a disruption of critical production machines or illness of employees, then specific business actors are hindered by failure of timely availability of necessary information. That leads to new disruptions and so on. Transparency of the model on causes and effects is of big importance. This second step in the way of working results in a table (cf. Table 3) in which for each information process all business actors which supply relevant original facts and all aggregate production banks from outside the organization boundary are set out.

Table 3. List of Source Actors per Required Information Function

Actor	No	Required Information Functions	Source Actor or Ext. Production Bank	Fact Kind/ Object Class
A01	1	Client data	APB01	Cl_name, Cl_company, Cl_standard_course
	2	Standard courses of EC	APB02	EC_standard_course
	3	The companies the EC works for	APB03	EC-company
	4	Number of attendees which are enrolled for standard course	A01	ENROLLMENT
	5	Maximum capacity for standard course	APB04	EC_max_capacity
A02		……..		
		……..		

Table 3 is added with a fifth column with the corresponding fact kinds and object classes. The fact kinds from outside the organization boundary are assigned to the fact banks APB01 until APB04. Table 3 gives us also the possibility for checking the completeness of the model. If important input data for a specific information process is missing, then it can be resolved in one of four different ways. Firstly, look if an existent business actor is able to deliver the needed fact or facts. Secondly, look for a missing business actor within an existing process which should deliver this fact. Thirdly, look for a missing business process in which a business actor should provide the missing fact, and fourthly, if all three cases do not help you to find the required input data within the scope of the organization, then the relevant data should be read from an external aggregate production bank.

The ISM of EC, that contains all interstrictions between actors and fact banks is combined with the IAM and is exhibited in Figure 5. The aggregated model is called the CM.

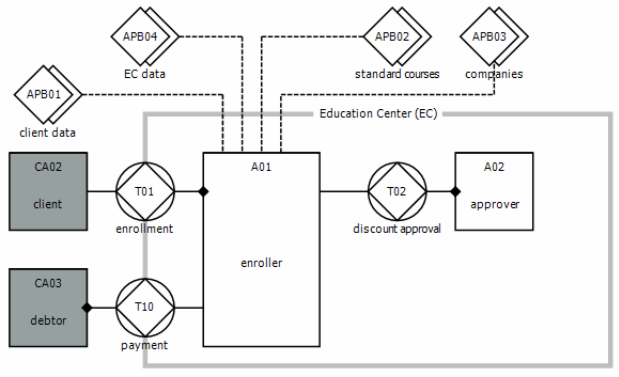


Fig. 5. The Construction Model of EC

3.4 Remembering by Business Actors

For each business actor have to be defined which fact kinds have to be remembered. Given the limits of the EC example we only have a limited number of fact kinds available. The Table 3, column 4 exhibits that we can restrict ourselves to business actor A01. A01 remembers the participants of the courses. All other facts come from external banks. From the fact bank T01 which is affiliated to A01 the number of registered participants per standard course can be determined.

Table 4. List of Fact Kinds per Business Actor/Fact Bank

Executor/ Bank	Description	Kind	Object 1	Object 2
A01/T01	ENROLLMENT	Object		
	cname	Unary fact	ENROLLMENT	
	ccompany	Binary fact	ENROLLMENT	COMPANY
	cstandard_course	Binary fact	ENROLLMENT	STANDARD_ COURSE
	[E] has been performed	Result		
A02/T02	Discount for [E] has been approved	Result		
EXT/APB02	STANDARD_COURSE	Object		
EXT/APB03	COMPANY	Object		
EXT/T10	[E] has been paid	Result		

Furthermore, we discussed that relevant client data from APB01 which is linked to the enrollment also should be remembered by A01. This is shown in Table 4. If this table is added with the result kind of each actor, then Table 4 becomes equal to the Bank Contents Table (BCT), known from Dietz [2]. Next, the FM which shows the declarative-shape business rules can easily be created from the BCT. See Figure 6.

In the previous text of this paper we saw that a business actor remembers data, i.e. objects, instances of fact kinds (unary and binary) and instances of result kinds (also called P-fact kinds). But, the question is now, when does the remembering of the mentioned data take place? Let us first look at the creation time of a production result

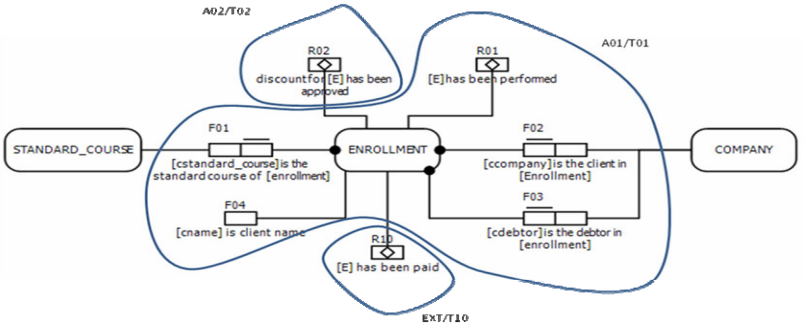


Fig. 6. The Fact Model of EC

(P-fact). The creation time of a P-fact is not equal to the execution time of the corresponding P-act because principally the execution time of a P-act is not precisely known.

It is chosen to equate it with the creation time of the corresponding C-fact 'stated'. Then, the creation time C-fact 'accepted' is chosen to be the actual creation time of the P-fact. From that time the P-fact starts to exist operationally. However, the explanation of the entity life cycle (cf. Fig. 7 gives motivation to model the remembering entities, its dependent facts and original properties in their prenatal phase, which is considered to coincide with the creation time C-fact requested. Once they exist in the production bank, the organization conceives their existence and is thus able to use (reproduce) them.

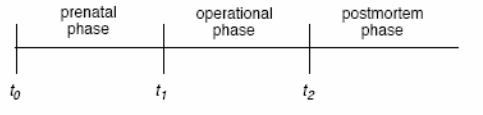


Fig. 7. Existence line of Entity

Back to the example of the EC, the approver may assume after a request for an approval that the enroller already is remembering specific client information. In this example, we have not elaborated the need for information of the approver. Although the P-fact of the enroller still not exists operationally, there would have undoubtedly gone an information link between A02 and the fact bank T01 in Figure 5 if we had elaborated the information need of A02.

3.5 Modeling Information Processes

The fourth step of the way of working concerns the design of the construction model of the information organization. Figure 5 exhibits that A01 puts their facts into the fact bank T01. One of the facts to remember is the unary fact client name. From now, A01 is liable for the correct client name linked to the enrollment. Assume that A02 would like to make its promise for performing its production act dependent of the client. Therefore, A02 has to reproduce the client name which is linked to the

enrollment and which has been remembered in T01. See the corresponding information link in Figure 8. The three layers in Figure 8 correspond with the three layers, business, infological and datalogical, in Figure 4.

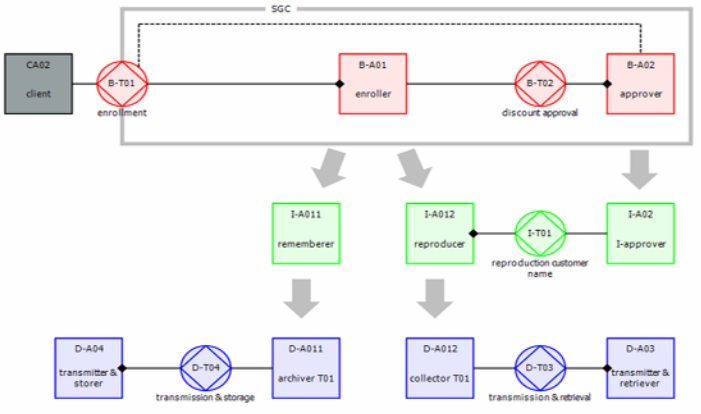


Fig. 8. Reproduction of an original Fact by any Business Actor

From the construction point of view the enroller B-A01 has both an informa ability I-A011 for remembering facts and an informa ability I-A012 for reproducing facts. The informa ability of the approver I-A02 requests the reproducer I-A012 for the original fact which is owned by B-A01. The forma ability D-A012 collects this fact from the fact bank T01. This picture shows that the business actor which remembers the original fact is responsible for the correctness of the information which is requested by another business actor. Figure 8 gives an illustration about how notions as responsibility as well as reusability of infological and datalogical production actions are modeled in the construction models of the I-organization and the D-organization, respectively. Look for a detailed elaboration of the way of modeling the information and documentation organization at the research paper of de Jong [14, 16]. In this section we will restrict ourselves in applying the theory of the mentioned paper using the EC example.

Information processes are modeled through the I-organization so that the B-actor only needs to focus on its informa ability for both remembering entities and reproducing information. Conforming to the description in the previous subsection, the prenatal phase of an entity in which remembering takes place normally coincides with the creation time of the C-fact requested. Besides that, I-transactions for the production of information are taken place during the execution of the action rules. Both kinds of information processes will be exhibited by the ontological model of the I-organization, the IAM of the I-organization, of EC. To build the I-IAM one has to determine all information functions per internal B-actor. The B-actor initiates as an external actor of the I-organization for each information function an I-transaction (cf. Fig. 9). Besides that, for every B-actor one I-transaction can be defined for remembering a fact to a particular production bank.

Table 5. Transaction Result Table of I-Interaction Model

Transaction kind	Result kind
I-T01 enrollment remembering	enrollment E has been remembered
I-T02 reproduction client data for new E	client data for new E has been reproduced from APB01
I-T03 reproduction standard course	standard course data has been reproduced from APB02
I-T04 reproduction company	company data has been reproduced from APB03
I-T05 #participants(SC) computation	#participants(SC) has been computed
I-T06 max_capacity reproduction	max_capacity has been reproduced from APB04
I-T07 enrollment E reproduction	enrollment E has been reproduced

Table 3 exhibits five information functions which can be linked to information processes. In addition to these processes, which reproduce original facts and perform infological actions on these original facts possibly, there should also be defined a remembering process that remembers the production facts of A01. So, in the I-IAM should be modeled six information processes which in total consist of seven transactions. Table 5 contains the transaction result table of the I-IAM of EC. The benefit of modeling functional requirements into I-transactions of the I-organization (instead of a direct modeling into an I-application model) is that the responsibility for those services is clearly shown. Without modeling the I-organization, the only visible responsibility is that of the B-actors. Therefore, the presence of the I-actors clearly shows the amount of the responsibility regarding the informational/intellectual services needed by the B-actors.

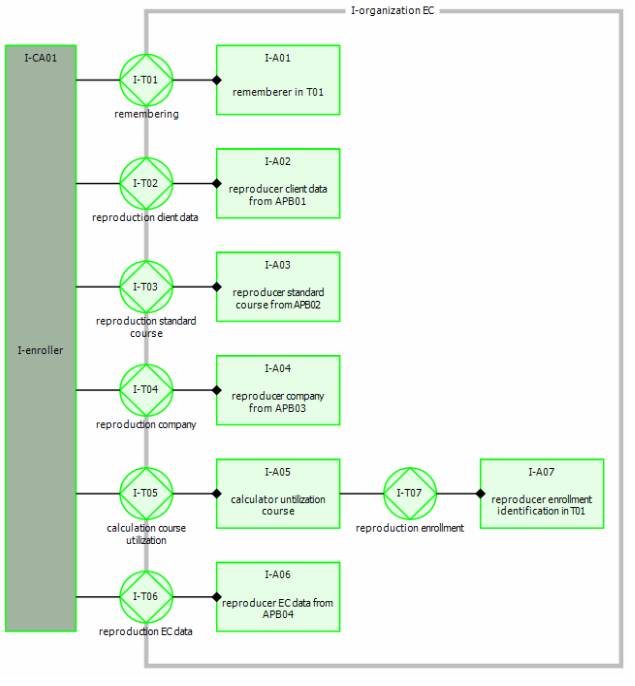


Fig. 9. The I-Interaction Model of EC

4 Conclusion and Further Research

This paper proposes a way of working for designing the information organization which is divided in four steps. The functional model of the information organization is derived from the construction model of the business organization. Subsequently, the information functions are analyzed to determine the original facts on which infological actions within the corresponding information processes are executed. Next, the required facts are bundled per business actor which remembers these facts and which is also responsible for the quality of each of these facts. Finally, the construction model of the information organization is designed for reusability purposes of the infological actions. This way of working results in the ultimate ontological model of the information organization which is the starting point for engineering the implementation model of the information organization.

References

1. Dietz, J.L.G., et al.: Enterprise Engineering: The concise Manifesto - version 9 (2010)
2. Dietz, J.L.G.: Enterprise Ontology – theory and methodology. Springer, Heidelberg (2006)
3. Dietz, J.L.G.: The deep structure of business processes. Communications of the ACM 49(5), 59–64 (2006)
4. Winograd, T., Flores, F.: Understanding Computers and Cognition: A New Foundation for Design. Ablex Corporation, Norwood (1986)
5. Dietz, J.L.G.: Generic recurrent patterns in business processes. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 200–215. Springer, Heidelberg (2003)
6. Dietz, J.L.G.: The Atoms, Molecules and Fibers of Organizations. Data and Knowledge Engineering 47, 301–325 (2003)
7. Van Reijswoud, V.E., Mulder, J.B.F., Dietz, J.L.G.: Speech Act Based Business Process and Information Modeling with DEMO. Information Systems Journal (1999)
8. Dietz, J.L.G.: Towards a LAP-based Information Paradigm. In: 9th International Working Conference on the Language Action Perspective on Communication Modelling, New Brunswick, New Jersey, USA (2004)
9. Austin, J.L.: How to do things with words. Harvard University Press, Cambridge (1962)
10. Searle, J.S.: Speech Acts, an Essay in the Philosophy of Language. Cambridge University Press, Cambridge (1969)
11. Habermas, J.: Theorie des Kommunikatives Handelns, Erster Band. Suhrkamp Verlag, Frankfurt am Main (1981)
12. van Reijswoud, V.E.: The Structure of Business Communication: Theory, model and application. Delft University of Technology, Delft (1996)
13. Mulder, J.B.F.: Rapid Enterprise Design. Technical University Delft, Delft (2006)
14. de Jong, J., Dietz, J.L.G.: Understanding the Realization of Organizations. In: Albani, A., Dietz, J.L.G. (eds.) CIAO! 2010. LNBIP, vol. 49, pp. 31–49. Springer, Heidelberg (2010)
15. Dietz, J.L.G.: Architecture, building Strategy into Design. Academic Service, The Hague (2008)
16. de Jong, J.: Integration Aspects between the B/I/D Organizations of the Enterprise. In: Albani, A., Barjis, J., Dietz, J.L.G. (eds.) CIAO! 2009. LNBIP, vol. 34, pp. 187–200. Springer, Heidelberg (2009)

Control Organization: A DEMO Based Specification and Extension

David Aveiro[1,2], António Rito Silva[2,3], and José Tribolet[2,3]

[1] Exact Sciences and Engineering Centre, University of Madeira, Caminho da Penteada 9000-390 Funchal, Portugal
[2] Center for Organizational Design and Engineering, INESC-INOV Rua Alves Redol 9, 1000-029 Lisboa, Portugal
[3] Department of Information Systems and Computer Science, Instituto Superior Técnico, Technical University of Lisbon
daveiro@uma.pt, {jose.tribolet,rito.silva}@ist.utl.pt

Abstract. In this paper we apply the Design and Engineering Methodology for Organizations (DEMO), to specify an ontological model for the generic Control Organization that we argue that exists in every organization. With our proposal, DEMO is extended so that we can specify critical properties of an organization – that we call *measures* – whose value must respect certain restrictions imposed by other properties of the organization – that we call *viability norms*. We can now also precisely specify defined *resilience strategies* that control and eliminate *dysfunctions* – violations of viability norms caused by *exceptions*. On top of this, we can also keep a systematic trace of the history of dysfunctions of an organization and of *control* acts executed to eliminate them. All of these facts are structured in a systematic manner and provided in a set of proposed tables, which are useful for a variety of purposes like (1) making control *responsibilities* clear and making organization agents *accountable* for bad control decisions, as well as (2) allowing *more informed organization change decisions*.

Keywords: enterprise engineering, control, DEMO, viability, dysfunction, exception handling.

1 Introduction

Our research focuses on improving the management of two related kinds of organizational change: *resilience change* and *microgenesis change*. Both kinds of changes are reactions to *dysfunctions,* i.e., deviations from what the *norm* is or from what would be expected [1]. An organization has to be prepared for timely and adequate responses to such dysfunctions so that its viability is maintained. To illustrate our ideas, we convey examples from the scenario of a library, introduced in [2] and extended in our research, as to better accommodate concepts we're proposing. The main activities of the library are *book loaning* and *offer book history courses*. We can define three norms: (1) *min average number of registrants in book history courses 1 week before start is 14*, (2) *min total income per month is 900€* and (3) *max loan declines per week is 30*. A possible dysfunction in the second norm is: *average number of registrants in*

A. Albani, J.L.G. Dietz, and J. Verelst (Eds.): EEWC 2011, LNBIP 79, pp. 16–30, 2011.

book history courses is 7 on March 23th 2009. This can be a very serious situation because, as a consequence, the library may lose income needed to acquire enough resources and eventually go bankrupt. Dysfunctions will have a *cause* which may be *expected* or *unexpected*. If the cause is expected, certain *resilience strategies* may already exist that can be activated to eliminate or circumvent dysfunctions [3], [1]. If the cause is unknown we will be in the presence of an *unexpected exception*. This unexpected exception will have to to be *handled* so that its concrete nature is detected and actions are undertaken that either eliminate or circumvent it, solving the dysfunction. The first time not enough students were registering it had, as a consequence, a dysfunction in the norm of *min total income per month*. As a result of this dysfunction, a handling process was initiated to detect the root cause (unexpected exception) of lack of income, namely: *lack of advertisement of courses*. The resilience strategy *distribute course fliers* was designed and chosen as solution to avoid the referred (previously unexpected) exception.

The handling of unexpected exceptions constitutes a central aspect of our research, namely *change* through the *(re)Generation*, *Operationalization* and *Discontinuation* of organizational artifacts which will eliminate or circumvent the determined cause of dysfunction. We consider an *organization artifact* (OA) as a construct of an organization like a business rule (e.g. "if invoice arrives, check list of expected items") or an actor role (e.g. library member). Change of OAs to handle dysfunctions is considered a special kind of dynamics that – inspired in philosophy literature on this subject – we call *microgenesis* [4]. We find that change is also driven by the detection of *opportunities of improvement* which will increase the viability of an organization and place it ahead of *competition* [5]. This is proactive change, as opposed to reactive change in the cases of resilience and microgenesis. The general focus of our research is on modeling the aspects of reactive change: (1) the resilience dynamics of strategies to solve known exceptions causing dysfunctions and (2) the microgenesis dynamics of handling unexpected exceptions also causing dysfunctions. This paper, in turn, focuses on the first aspect: resilience dynamics. In section 2 we develop our research problem and related work. Section 3 presents our application of DEMO, in order to specify and handle resilience dynamics of organizations. Section 4 concludes this paper with a critical review on work done and future lines of research.

2 Problem, Motivation and Related Work

Above findings helped us to trace down and focus on the following research problem: an *absence of concepts and method for explicit capture, and management of information of exceptions and their handling; this includes the design and selection of OAs that solve dysfunctions caused by exceptions*. Not immediately capturing this handling and the consequent resulting changes in reality and the model of reality itself, will result that, *as time passes, the organization will be less aware of itself than it should be*, when facing the need of future change due to other unexpected exceptions. We focus on this problem in the context of small timely changes, as opposed to large impact changes in the context of IT/IS projects, mergers, acquisitions and splittings of organizations. This problem is highly relevant since, as we saw in the library examples, dysfunctions like lack of enough income can compromise a whole organization.

Also, exception handling can sometimes take almost half of the total working time, and the handling of, and recovering from, exceptions is expensive [6]. *What causes can be identified relating to this problem?* We identify what seems to be a lack of capture and management of relevant information of past unknown exceptions and their handling. Many events (which were previously unknown exceptions) can have already been known or expected in the past, but can be (frequently) forgotten and become again unexpected (unknown) due to: (1) absence of explicit representation of (i) specific exceptions and actions that were executed (in an Ad hoc and unstructured way) for their handling and (ii) engineered OAs to solve them [7] or (2) removal of human agents from a certain organizational actor role which had established and tacitly memorized specific (informal) rules to handle specific exceptions occurring in such actor role [6]. The shortcoming of lack of continuous update of models aligned with the continuous change of reality has been addressed, by and large, in research and practice in the context of Workflow Management Systems (WfMS) – see, for example, [7] and [8]. However, current solutions assume that an organization will be using a WfMS, which will not be the case of many organizations. And, even in the case of organizations using WfMS, relevant activities may happen outside of IT context and we may also want to address exceptions related to them. From Complex Adaptive Systems (CAS) literature, [3] (p. 33) and philosophy [1], we find that systems maintain an internal model of the world (of themselves and the environment) so that they can activate specific resilience strategies to react, appropriately and in time, to certain known exceptions or fluctuations in critical norms that guarantee the system's viability. We also find that a system adapts with incremental changes [4], having as a main purpose to survive and evolve among competition, by having credit mechanisms which favor changes (adaptations) that increase the system's viability and constitute criteria of measuring success [3] (p. 34), [9] (p. 5). One of the premises from CAS theory is that, to solve new exceptions, “rule pieces” that constitute current resilience strategies that solve similar exceptions may be re-utilized to build new resilience strategies or new organization artifacts to solve the new exceptions. From unexpected exception handling in WfMS [7] and ODE [10], we find that information on the history of organization change is an essential asset in moments where change is again needed, i.e., in microgenesis dynamics.

Specifying resilience and microgenesis dynamics and keeping a systematic history of their execution is deemed as a solution to our general research problem, so that exception handling and organization change is more efficient and effective. Microgenesis is the main focus of our general research project, but to precisely specify its dynamics we need to precisely specify resilience dynamics – the focus of this paper. To ground our solution, we decided to narrow our research focus, choosing a particular OE approach, namely, the Design & Engineering Methodology for Organizations (DEMO) [2]. From several approaches to support OE being proposed, DEMO seems to be one of the most coherent, comprehensive, consistent and concise [2]. It has shown to be useful in a number of applications, from small to large scale organizations – see, for example, [11] and [12] (p. 39). Nevertheless, DEMO suffers from the same general OE shortcomings referred above. Namely, DEMO models have been mostly used to devise blueprints to serve as instruments for discussion of broader scale organizational change or development/change of IT systems [12] (p. 58) and does not, yet, provide modeling constructs and a method for a continuous update of its

models as reality changes, driven by exceptions (microgenesis) nor for the continuous control (resilience) that we need to exert on organizations to guarantee viability.

Contributions of our research are presented in the next sections. We apply DEMO, to devise a set of concepts and a method that systematically address the elicited problem in the resilience aspect. While proceeding, the reader which is unfamiliar with this methodology is advised to also consult [2] or [11] or other publications in: www.demo.nl.

3 Applying DEMO to Specify the Control Organization

Microgenesis change starts in a particular context in resilience dynamics, namely, when a certain dysfunction cannot be solved by any existing resilience strategy. Hence, the need of focusing first in resilience, which is one of the aspects of control of an organization. The study of control is a vast area and in this paper we address only control issues directly related with our main focus, that is, microgenesis change. Namely, we will now look in detail how resilience dynamics of an organization can be precisely and coherently systematized. We leave the study of other control issues as future research work. We do not find the aspect of resilience dynamics – in the form of specification of viability and its control – addressed in current version of DEMO. One of the contributions of our research is to apply DEMO to model what we propose to call the Control Organization (CO). The CO's ontological model is the conceptualization of a generic organization considered to exist in every organization and responsible for controlling its viability. We next present the formulation of the main aspect models of the CO. In practice, the CO's ontological model will be a default subset of the ontological model of every organization. But for separation of concerns reasons we model the CO as a "separate" organization although, in practice, it is included in the controlled organization itself. The following subsections portray, in an intertwined manner, the CO's aspect models while having, as anchors, a number of tables we propose to express state information of the control aspect of an organization. The CO's State Model is formulated, in World Ontology Specification Language (WOSL) [13], in Figure 2, which depicts the CO's State Space Diagram (CO's SSD). WOSL is highly based in the ORM fact oriented modeling language [14], extending it with the ability of modeling events (result kinds) affecting facts. Further ahead we can also find the formulation of the CO's Construction Model (CO's CM) – depicted in the CO's Organization Construction Diagram (CO's OCD) – and the CO's Action Model (CO's AM) – specified in a set of three generic action rules.

3.1 Measures and Viability Norms Table

Logically, the viability of an organization should be specified by the declaration of viability norms for measures related with its production banks. This makes sense because, inevitably, viability of an organization is related with its production. Namely, for an organization to be viable, it needs to be able to satisfy environmental demand and expectations, with appropriate quality, while acquiring the appropriate amount of resources. We find that the Object Property List (OPL) – part of DEMO's State Model – is a convenient way of specifying fact types that are proper (mathematical) functions, and of which the range is a set of values [2]. The fact types in an OPL are

called properties (of object classes). We observe, from the OPL of the library scenario, that certain properties specify restrictions on another property. E.g., we have property *max_copies_in_loan* which specifies the restriction of the maximum number of book copies a certain member of the library is allowed to have in loan. Observance of this restriction is verified in the action rule that decides on an acceptance or a decline in transaction *loan start*. It logically follows that, in the SM of a certain organization, certain properties will have to be declared that specify restrictions on certain measures related to its information banks (i.e., to its production and coordination information). Taking the case of the library, one of its production banks is PB01, also named as *membership fee payments*. To make sure one is able to cover all expenses of the library, we will need to measure a relevant property related with PB01, namely, *total income per month*. As a viability norm, we can declare a certain necessary minimum income per month. We do this with property *min total income per month*. To detect possible problems in loans, we can measure a relevant property related with coordination bank CB04 of transaction *loan start*, namely, *loan declines per week*. As a viability norm, we can declare an acceptable maximum amount of declines per week. We do this with property *max total loan declines per week*. We depict, in Figure 1, part 1 of the CO's SSD which expresses our reasoning so far. Object class MEASURE represents the population of all properties of the main organization which are measures related with information banks. In other words, it represents the subset of the properties of an organization which constitute measures that will be repeatedly observed (measured) for viability control ends. Certain other properties of an organization will specify restrictions on the properties that are measures, so that viability is assured. Object class VIABILITY NORM represents the population of all these restriction properties. Besides knowing which properties of the main organization are measures and viability norms, we need to know explicitly which are the viability norms imposed on each measure. We model this need with the fact type, explained by predicative sentence: *[viability norm] restricts [measure]*.

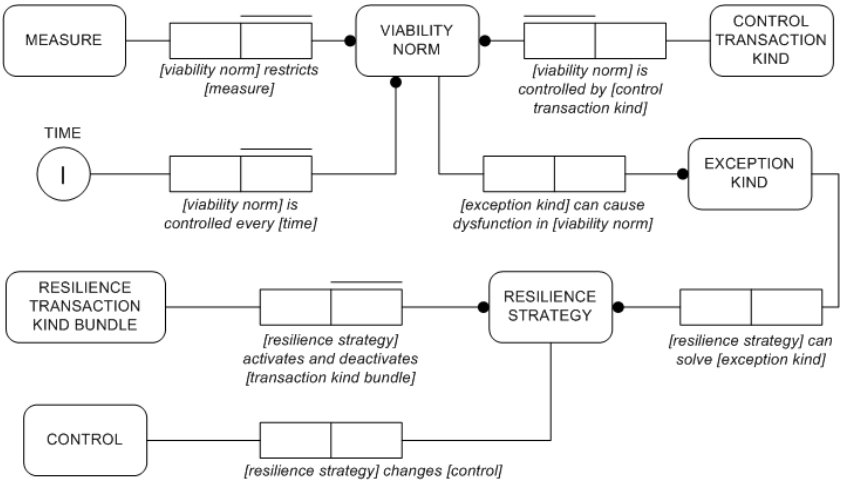


Fig. 1. Control Organization's SSD - part 1

We propose the Measures and Viability Norms Table (MVNT) to express this aspect of the CO's SM, exemplified in Table 1 for the case of the library. The practical relevance of this table is a quick and summarized glance on which are the critical variables – measures – on which to evaluate an organization's viability and what restrictions – viability norms – apply for them so that one can determine if dysfunctions are happening or not. In other words, it is a clear and succinct specification of what is the normal and allowed quality of operation for the relevant production of an organization. As it will be clear ahead with the examples we provide, in situations of occurrence of expected exceptions or necessity of change, information of these measures and restrictions on them is essential to assess change impact and eventually adopt new values for such restrictions.

Table 1. Measures and Viability Norms Table of the library

measure	**viability norm**	**scale**
total income per month	min total income per month	EURO
loan declines per week	max loan declines per week	NUMBER
average # of registrants in book history courses 1 week before start	min average # of registrants in book history courses 1 week before start	NUMBER

All measures will be properties that are derived fact types and whose derivation rules imply a certain computation on facts of the related information bank. For the examples in Table the derivation rules are quite simple and implicit in the property name so we skip from presenting them. But, depending on the complexity of the viability measure and/or related information banks, derivation rules may have to be specified, similarly to what is done for the case of the OPL of the library.

3.2 Control Responsibilities Table

To guarantee an organization's viability, control acts will have to be executed with a certain frequency. In such acts, measures will be measured, and evaluated, against their respective viability norm. In case the norm is respected, it means such norm is in a state of *function* and nothing else needs to be done. In case the norm is violated it means the norm is in a state of *dysfunction* and other control acts will have to be executed to solve such dysfunction, namely, initiation of resilience or microgenesis change. Following the distinction axiom [2], the acts of measuring and evaluating measures against viability norms are purely infological and, thus, are not part of our model. The acts of deciding on certain control acts as a reaction to a dysfunction are ontological and will be described in detail next.

Table 2. Control Responsibilities Table of the library

viability norm	**transaction kind**	**time**	**responsible actor role**
min total income per month	T20 - finance management	month	A20 - finance manager
max loan declines per week	T21 - general management	week	A21 - director
min average # of registrants in book history courses 1 week before start	T22 - course management	week	A21 - director

It logically follows that certain transaction kinds of an organization will control certain viability norms and, as such, certain actor roles, will be responsible for carrying out the respective control acts. Additionally, it is essential to specify the frequency with which such control is to be done. These relationships are depicted in the CO's SSD in Figure 1, with fact types explained by the following predicative sentences: *[viability norm] is controlled by [control transaction kind]*; *[viability norm] is controlled every [time]*. Transaction kinds of the Construction Model of the main organization which exert control are captured in object class CONTROL TRANSACTION KIND. We propose another table to express this other aspect of the CO's SM namely, the Control Responsibilities Table (CRT), for the case of the library, found in Table 2. T20, T21 and T22 are self-activation transaction kinds that execute control on the associated viability norms with the specified frequency. The practical relevance of this table is to clearly express two dimensions of responsibility of control of viability norms, namely *who* and *when*. This will be very helpful for auditing ends in quickly and clearly identifying responsibilities in case dysfunctions happen and inappropriate decisions were taken regarding control acts.

3.3 Exceptions and Resilience Strategies Table

We propose object class EXCEPTION KIND – also depicted in Figure 1 – so that we can specify expected exception kinds. We will need to relate each exception kind with a viability norm which can be in a dysfunction state due to occurrences of such exception kind. This relation is specified by the fact type: *[exception kind] can cause dysfunction in [viability norm]*. We propose also object class RESILIENCE STRATEGY, so that we can specify resilience strategies that can solve expected exception kinds. We propose another table to express this other aspect of the CO's SM, namely, the Exceptions and Resilience Strategies Table (ERST), exemplified, for the library scenario, in Table 3. The practical relevance of the ERST is to provide a comprehensive and summarized view of which exceptions exist that cause dysfunction on an organization's viability norms and which resilience strategies can be activated to solve such exceptions and eliminate the dysfunction. When an expected exception occurs, the several alternatives that exist to solve it are easily accessible so that the authorized controller can decide on the more adequate choice. The ERST will also provide useful information in change context where it may become clear that certain exceptions have become obsolete. In this case, these exceptions may be discontinued, along with the associated resilience strategies in case they do not solve any other exception. The ERST provides very useful information for the process that we propose to call as *organization artifact garbage collection*. This name is inspired in the method of memory garbage collection from computer science. In any organization it is natural that organization artifacts become obsolete, in the sense that they are not useful or used anymore. An example is a resilience strategy that solves an exception that will certainly not occur again. That strategy can be discontinued as it will not be executed anymore. So, whenever a change process is finishing in any organization or, at certain time i ntervals (e.g., weekly or monthly), the organization artifact garbage collection process – an inherently human activity – should verify if any exception has become obsolete and discontinue it, eventually along with any associated resilience strategies. The ERST will be a sound starting point and aid for such process.

For space reasons, we leave out of this paper another table we propose: the Resilience Strategies Definition Table (RSDT), which details how dynamics of these

Table 3. Exceptions and Resilience Strategies Table of the library

viability norm	exception kind	resilience strategy
VN01 - max loan declines per week	E01 - abnormal high rate of loan requests due to exams season	RS01 - increase value of max_copies_in_loan
VN02 - min average # of registrants in lang. history courses 1 week before start	E02 - lack of advertisement of courses	RS02 - distribute course fliers
	E03 - general lack of interest in courses	RS03 - delay courses start
		RS04 - reduce # of courses

strategies are realized in terms of other transactions and controls. The practical relevance of the RSDT is to provide a comprehensive view of details of each resilience strategy and, in the context of microgenesis change, to provide, along with the ERST, clues on how to solve new previously unexpected exceptions. The RSDT and ERST follow the premise from CAS theory that, to solve new exceptions, "pieces" that constitute current strategies that solve similar exceptions may be re-utilized to build new resilience strategies or new organization artifacts to solve these new exceptions.

3.4 Dysfunctions Table

We will now focus on the dynamic aspect of the CO, centered around the observation of dysfunction and appropriate reactions to them. Figure 2 presents part 2 of the

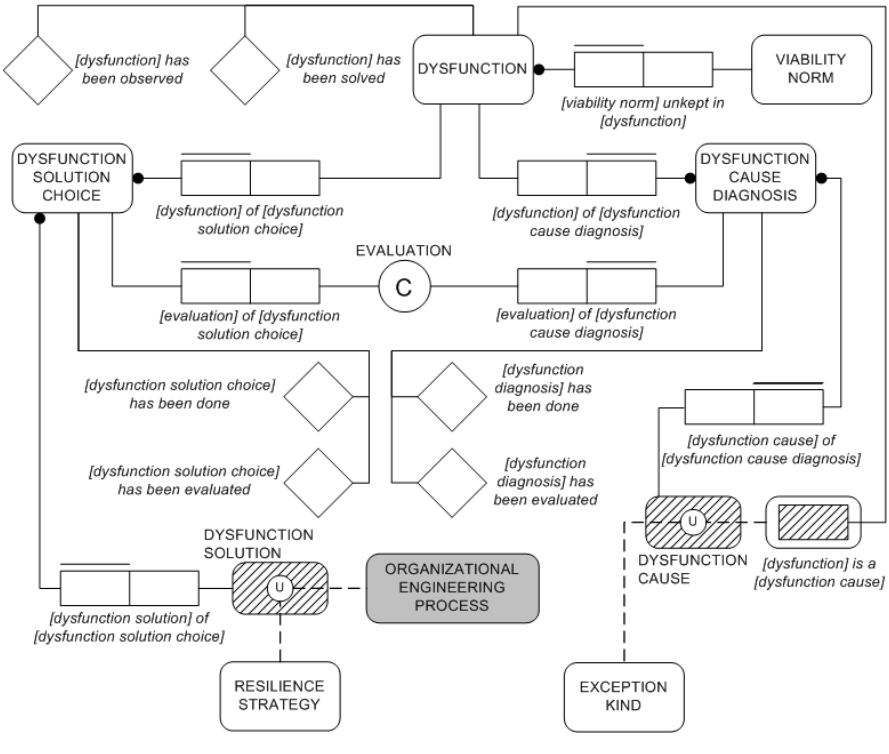


Fig. 2. Control Organization's SSD - part 2

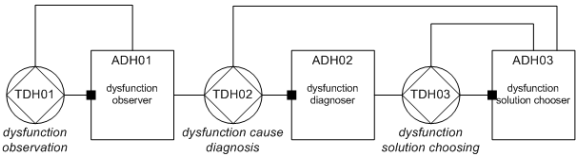


Fig. 3. Control Organization's OCD

Control Organization's SSD, expressing what will be relevant state information of dynamics of the control aspect of an organization. We identify a generic pattern of three transaction kinds. These are considered to be implicitly enclosed by what we proposed to call the *generic control transaction kind*. They are associated with their respective actor roles which are considered to be implicitly part of the *generic control actor role*. Together they form the generic CO's OCD, depicted in Figure 3. This means that each transaction kind belonging to the main organization's CM and specified, in the CO, as controlling a certain viability norm will, together with the respective actor role responsible for its execution, implicitly adopt the behavior specified in the CO's OCD. These three transaction kinds constitute what we can call the *dysfunction handling process* (DHP) and, together with their respective actor roles specify a pattern of behavior that underlies every control transaction kind. Normally, it will be the case that the actor role responsible for executing a certain control transaction can be considered as a composite actor role aggregating the three actor roles defined in the CO's OCD. But it can also be the case that there is delegation of these actor roles to other actor roles, depending on the volume of work needed to be done or the needed expertise. Thus, we can say that each viability norm has, associated to it, a particular (mini) Control Organization corresponding to the responsible actor role that inherits the behavior specified by the CO's OCD. This CO will be responsible for the process of handling dysfunctions in this particular viability norm. The *dysfunction handling process* (DHP) process starts with transaction kind *dysfunction observation*, with the associated result kind, *dysfunction has been observed*. The execution of this transaction may imply the creation of a dysfunction's first constituent fact, instances of the fact type explained by predicative sentence: *[viability norm] is unkept in [dysfunction]*. Instances of this fact type, together with other information derived from the appropriate sources, will populate our proposal of a Dysfunctions Table (DFT), expressed in Table 4.

Table 4. Dysfunctions Table of the library

viability norm		**measurement**			**dysfunction**	**solved**	**solution**
min total income per month	900 €	total income per month	805 €	Jun 15 2008	DF01	Jun 21 2008	OEP01
min average # of registrants in book history courses 1 week before start	14	average # of registrants in book history courses 1 week before start	9	Sep 12 2008	DF02	Sep 26 2009	OEP02
			11	Jan 4 2009	DF03	Feb 1 2009	OEP03
			10	Feb 8 2009	DF04	Feb 15 2009	RS02
			8	Mar 23 2009	DF06	Apr 7 2009	OEP04
			7	Apr 18 2009	DF07	May 14 2009	RS04
max loan declines per week	30	loan declines per week	38	Feb 8 - Feb 15 2009	DF05	Feb 20 2009	RS01

The DFT – or a similar table, adaptable to an organization's needs – will provide a summary of current (unsolved) dysfunctions and past (solved) dysfunctions. This will be useful, for example, in diagnosing a recurrent previously unexpected exception. It will also be an instrument for the "controllers of the controllers" so that higher hierarchy in the organization can act if certain dysfunctions are not being appropriately handled by their responsible controllers.

3.5 Dysfunction Diagnosis and Actions Table

We will now go through the CO's OCD and AM, which will lead us to the explanation of the remaining object classes and fact types of the OC's SM, as well as the declared result kinds. These are the base for the last table of our current proposal for expressing the control aspect of an organization. We propose actor role *dysfunction observer*, to be responsible for detecting and reacting to dysfunctions, having the single action rule (part of the CO's AM):

```
on requested TDH01 (D)
              if <VN is violated> ≥
                viability_norm_of(D) = VN
                DCS = all DC in DYSFUNCTION CAUSE where
                  <DC can cause dysfunction in VN>
                if <DCS empty> ≥ <initiate OEP (D)>
                   € not <DCS empty>     request TDH02 (D, DCS)
                fi
              fi
no
```

This specifies that, in case a measurement is violating the viability norm (VN), a new dysfunction (D) is created that refers to VN. Then a list (DCS) is created of all possible causes that can cause VN being unkept, i.e., in dysfunction. If there is no violation nothing happens. If DCS is empty it means that no exception is known that causes such dysfunction – and, consequently, there are no defined resilience strategies to deal with this dysfunction – and it is necessary to initiate an Organizational Engineering Process (OEP) so that microgenesis dynamics will generate new OAs to cope with the (unexpected) exception causing dysfunction. Let's consider, for example purposes, the ERST in Table 3, the DFT in Table 4 and the date of Sep 12 2008. Resilience strategies to solve exceptions in norm VN02 – *min average # of registrants in book history courses 1 week before start* had not yet been generated, neither the exceptions themselves. In this case, this action rule would immediately initiate an OEP as DCS would be empty. If DCS is not empty, it means that one of the causes in DCS can be diagnosed as the cause of dysfunction D. Hence, transaction TDH02 – *dysfunction cause diagnosis* is requested, which uses information of D and DCS.

Now considering again Table 4 and the date of Mar 23 2009, both exceptions causing dysfunctions in VN02 are a possible expected cause so list DCS would have these two exception kinds as its elements. Result kind *[dysfunction diagnosis] has been done* is the result of TDH02, whose execution will imply the creation of the constituent facts of an instance of class DYSFUNCTION CAUSE DIAGNOSIS. Such instance basically specifies a decision on a possible cause of a certain dysfunction. The population of all possible causes is specified by object class DYSFUNCTION CAUSE which is a generalization (union of members) of object class EXCEPTION

KIND and derived fact type: *[dysfunction] is a [dysfunction cause]*. This simply means that, either another occurring dysfunction, or an exception kind, can be a dysfunction cause. Actor role *dysfunction diagnoser* is responsible for execution of TDH02 having action rule:

```
on requested TDH02 (D, DCS)
  if not <DCS empty> ≥
    dysfunction_cause_of(new DCD) =
    <choose an adequate DC in DCS>
    dysfunction_of(DCD) = D
    RSS = all RS in RESILIENCE STRATEGY where <RS solves exception_kind_of(DC)>
    DCS = DCS less DC
    request TDH03 (D, DCS, DCD, RSS)
    € <DCS empty> ≥ <initiate OEP (D)>
  fi
no
```

This specifies that, just like in the previous action rule, an OEP is initiated if the list of dysfunction causes is empty. In this action rule this means that all dysfunction causes and respective resilience strategies will have been already tried (after successive diagnoses and respective solution trials) and the dysfunction was not solved. This is the case of the particular DHP handling dysfunction DF06 (c.f. DFT table of the library). In a certain point in the execution of this DHP, after resilience strategies RS02 and RS03 were activated and the dysfunction was not solved, OEP04 was initiated to solve DF06, which lead to the generation and operationalization of resilience strategy RS04. In the case that DCS has at least one cause to choose from, a new instance of dysfunction cause diagnosis (DCD) is created and a certain dysfunction cause (DC) is chosen to be the cause in the DCD. Considering the DHP of DF06, the first time the action rule currently being explained is executed, DCS consists in exceptions (E02, E03) and E02 is chosen as the cause. After this, a list (RSS) is created with all resilience strategies that can solve the exception kind associated with the diagnosed dysfunction cause (DC). In our example, at this stage, RSS will consist in resilience strategy RS02. Then, the chosen cause (DC) is removed from the list DCS, so that, if this action rule (diagnosis) is again executed, only one of the remaining causes can be chosen for a new iteration of resilience trials or no causes will remain in the list – the case that an OEP will be initiated. Finally, transaction TFH03 – *dysfunction solution choosing* is requested, which receives D, DCS, DCD and RSS.

Result kind *[dysfunction solution choice] has been done* is the result of TDH03, whose execution will imply the creation of the constituent facts of an instance of class DYSFUNCTION SOLUTION CHOICE. Such instance basically specifies chosen solutions for certain dysfunctions. The population of all allowed solutions is specified by object class DYSFUNCTION SOLUTION which is a generalization (union of members) of classes RESILIENCE STRATEGY and ORGANIZATIONAL ENGINEERING PROCESS. This means that either an existing resilience strategy or an organizational engineering process will constitute a particular dysfunction's solution choice. Actor role *dysfunction solution chooser* is responsible for execution of TDH03 having action rule:

```
on requested TDH03 (D, DCS, DCD, RSS)
              if not <RSS empty> ≥
                CRS = <choose an adequate RS in RSS>
                dysfunction_solution_of(new DSC) = CRS
                dysfunction_of(DSC) = D
                <activate CRS for an adequate time period T>
                <wait T> && <deactivate CRS>
                if <CRS solved dysfunction> ≥
                  evaluation_of(DSC)="good"; evaluation_of(DCD)="good"
                  state_of_(D)="solved"
                € not <CRS solved dysfunction> ≥
                  evaluation_of(DSC)="bad"; RSS = RSS less CRS
                  request TDH03 (D, DCS, DCD, RSS)
                fi
              € <RSS empty> ≥
                evaluation_of(DCD)="bad"
                request TDH02 (D, DCS)
              fi
no
```

This action rule begins with the choice of a resilience strategy (CRS) from the received list RSS – containing all RSs for the diagnosed dysfunction cause. In our example of the DHP handling DF06, this action rule will be executed two times, one with RSS having just RS02 and another having just RS03. Then, the constituent facts of a new dysfunction solution choice (DSC) are created. After this, CRS (in our example, RS02 and, in the next iteration, RS03) is activated for a certain amount of time chosen to be adequate for the solving process, after which it is deactivated. Then, if the dysfunction got solved both choices of dysfunction cause and dysfunction solution are evaluated as "good" and dysfunction is considered as solved, corresponding to the occurrence of instances of the three result types: *[dysfunction diagnosis] has been evaluated*; *[dysfunction solution choice] has been evaluated*; *[dysfunction] has been solved*. An example of such situation is the handling of DF04 specified, ahead, in Table 5. If the dysfunction was not solved with the chosen CRS, then such choice is evaluated as "bad", with the creation of an instance of the second result kind above. This same CRS is removed from the list of possible solutions and there is a recursive request for TDH03 with the same parameters (with RSS having one less element). An example of execution of this step is the handling of DF07 – also specified in table 5 – when resilience strategy *delay courses start* was evaluated as a bad solution and there was still the option in RSS of: *reduce number of classes*.

Table 5. Dysfunction Diagnosis and Actions Table of the library

dys-function	observed	chosen cause	choice eval	chosen solution	choice eval	solved
DF03	Jan 4 2009	lack of advertisement of courses	bad	distribute course fliers	bad	Feb 1 2009
DF04	Feb 8 2009	lack of advertisement of courses	bad	distribute course fliers	bad	Feb 15 2009
		general lack of interest in courses	good	delay courses start	good	
DF05	Feb 8 - Feb 15 2009	abnormal high rate of loan requests due to exams season	good	increase value of max_copies_in_loan	good	Feb 20 2009
DF06	Mar 23 2009	lack of advertisement of courses	bad	distribute course fliers	bad	Apr 7 2009
		general lack of interest in courses	bad	delay courses start	bad	
DF07	Apr 18 2009	lack of advertisement of courses	bad	distribute course fliers	bad	May 14 2009
		general lack of interest in courses	good	delay courses start	bad	
				reduce number of classes	good	

If it happens that RSS is empty, it means that all possible solutions for the chosen dysfunction cause have already been tried and evaluated as a "bad" choice, which means that the diagnosis should also be evaluated as "bad" and a new request for TDH02 has to be done so that another cause can be diagnosed. This is the case in our example where, in the handling of DF06, RS02 – distribute course fliers – was unable to solve the dysfunction and TDH02 had to be again requested for another cause to be diagnosed. This is what happened just next with the choice of cause *general lack of interest in courses*. These examples which are instances of the above proposed fact and result types will populate our proposal of a Dysfunctions Diagnosis and Actions Table (DDAT) expressed, for the case of the library, in Table 5. The DDAT – or a similar table, adaptable to an organization's needs – will provide a summary of the history of choices of dysfunction diagnosis and solutions. Such history corresponds to a trace of the execution flow of the three action rules of the CO or, in other words, a trace of the execution flow of each DHP. Hence, together with the DFT and CRT, this table provides a succinct and thorough trace of relevant control decisions of an organization, so that adequate measures can be taken against irresponsible agents, in a justified and detailed manner. Counting the good and bad choices of diagnosis and resilience strategies is a way to implement another premise from CAS, namely, a scoring mechanism which can help on better deciding which resilience strategies to associate with new (previously unexpected) exceptions or on the generation of new resilience strategies.

4 Conclusions

It was not trivial to arrive at the proposed structure of the CO. Following Design Research guidelines in [15], an intertwined and iterative simultaneous devising of the CM, AM and SM was undertaken to stabilize on a coherent and succinct model for the CO. Obviously this model has much room for improvement and other issues can and should be explored and researched regarding the CO. For example, rules can be declared that limit the allowed amount of variation of control of a certain property. We could, for example, define properties: *max allowed max copies in loan* and *min allowed max copies in loan* that would impose maximum and minimum values on the (control) property *max copies in loan* of the library. In this manner, resilience strategy *increase value of max copies in loan* would not be able to increase such value indefinitely. Another issue to explore is the responsibilities hierarchy. One can also declare measures and viability norms related to control transactions and also on transactions part of resilience strategies. All declared responsibilities regarding measures and viability norms will have to relate with responsibilities belonging to higher authority and, ultimately, with the supreme controlling authority of an organization – e.g., its administration board. Following a systematic and neutral approach of defining measures of an organization, their respective viability norms and the network of responsibilities relating them will be an aid in discussions of organizational structure (e.g. departments, boards, etc.)

We have focused on the portions of the CO specifying relevant issues of resilience change related with our main focus, which is microgenesis change. Developing and perfecting further aspects like the above example of control limits and hierarchical

structure are left as future research work. To our knowledge, issues addressed in the CO have been target of ample research and study – e.g.: on themes such as the *Viable System Model* (VSM) [16] and *Autopoiesis* [17] – but without sound formal and theoretical grounds such as the ones given by enterprise ontology. In fact, our proposal of the CO seems to fit in part of Beer's System 3 (also named *control*) of the VSM. We end up formalizing part of his intellectual contribution, specifying with a concise and coherent set of facts, how organizations are systems that maintain viability and have recursion control mechanisms. Our research builds on the existing and proven methodological framework of DEMO and its underlying theory. In a bottom up fashion, and following the important principles of *separation of concerns* and *verification by instantiation* [18], we use simple but comprehensive and relevant concepts from biology, philosophy and CAS for the formulation of the CO. Our proposal of the CO's ontological model is, at the same time, an application and an extension of DEMO, so that this method now has steps to address the specification of an organization's resilience aspect, both in the static and dynamic realms. Each major step is confined in each of the tables we propose. First, measures can be explicitly declared which, together with the associated viability norms, function as "sensors" of an organization's health, i.e., its viability. This is done in the MVNT. Second, important responsibilities are now thoroughly explicit and traceable: we can clearly know *who* is responsible (and *when*) for controlling each measure, as to guarantee the maintenance of each viability norm. This is specified in the CRT. Third, the ERST systematizes solutions – resilience strategies – for expected exceptions, an important tool for the controller to be more able to choose the best course of action in the context of a dysfunction. Entering the dynamic realm, the DFT and DDAT keep a record of all observed dysfunctions and the respective diagnose and solution actions performed to eliminate them. Evaluations of which were the good and the bad choices is also done. The facts systematized in these two tables constitute valuable information for controlling the controllers and for microgenesis change contexts. Concluding, with the facts we propose to express the control aspect, we have a thorough and precise way to formally express organizational knowledge in terms of (1) the control system of an organization and (2) the control history in terms of dysfunctions, known cause diagnosis and resilience decisions. Such knowledge is a central asset for humans handling resilience and microgenesis dynamics in a more informed and effective way.

References

1. Christensen, W.D., Bickhard, M.H.: The process dynamics of normative function. The Monist. 85, 3–29 (2002)
2. Dietz, J.L.G.: Enterprise ontology: theory and methodology. Springer, New York (2006)
3. Holland, J.H.: Hidden order: how adaptation builds complexity. Basic Books, New York (1996)
4. Bickhard, M.H.: Error dynamics: the dynamic emergence of error avoidance and error vicariants. Journal of Experimental & Theoretical Artificial Intelligence 13, 199–209 (2001)
5. Brown, S.L., Eisenhardt, K.M.: Competing on the edge: strategy as structured chaos. Harvard Business School Press, Boston (1998)
6. Saastamoinen, H., White, G.M.: On handling exceptions. In: Proceedings of Conference on Organizational Computing Systems, pp. 302–310. ACM, New York (1995)

7. Mourão, H.: Supporting effective unexpected exceptions handling in workflow management systems within organizational contexts. Science Faculty of Lisbon University (2007)
8. Casati, F., Pozzi, G.: Modeling exceptional behaviors in commercial workflow management systems. In: Proceedings of the Fourth CoopIS - International Conference on Cooperative Information Systems, pp. 127–138. IEEE Computer Society, Washington, DC, USA (1999)
9. Axelrod, R., Cohen, M.D.: Harnessing complexity: organizational implications of a scientific frontier. Basic Books, New York (2001)
10. Magalhães, R., Silva, A.R.: Organizational design and engineering (ode) - ode white paper - version 1 (2009)
11. Dietz, J.L.G., Albani, A.: Basic notions regarding business processes and supporting information systems. Requirements Engineering 10, 175–183 (2005)
12. Op' t Land, M.: Applying architecture and ontology to the splitting and allying of enterprises. TU Delft (2008)
13. Dietz, J.L.G.: A world ontology specification language. In: Chung, S., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 688–699. Springer, Heidelberg (2005)
14. Halpin, T.: Object role modeling: an overview. white paper (1998), `http://www.orm.net`
15. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Quarterly 28, 75–106 (2004)
16. Beer, S.: Brain of the firm: a development in management cybernetics. Herder and Herder (1972)
17. Maturana, H.R., Varela, F.J.: Autopoiesis and cognition: the realization of the living. D. Reidel Pub. Co (1980)
18. Dietz, J.: Is it $\varphi\tau\psi$ or bullshit? - farewell speech. Faculty of Electrical Engineering, Mathematics and Computer Science. Technical University of Delft (2009)

Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems

Marien R. Krouwel[1] and Martin Op 't Land[1,2]

[1] Capgemini Netherlands, P.O. Box 2575, 3500 GN Utrecht, The Netherlands
{marien.krouwel,martin.optland}@capgemini.com
[2] Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands

Abstract. Our research aims at finding key concepts to link agile enterprises with agile automated information systems. To effectively respond to environmental changes, such as in market needs, technology, regulations or law, enterprises need to be able to change their supporting information system(s) accordingly. The Design and Engineering Methodology for Organizations (DEMO) has already proven to be an effective tool in designing and realizing agile organizations. The Normalized System (NS) approach, on the other hand, has proven to be the key for developing agile Information and Communication Technology (ICT) systems, which support such agile organizations.

We found that DEMO and its underlying PSI-theory, and the principles and elements of the Normalized Systems approach match. Also we designed, using two cases of Dutch governmental subsidy schemes, a few simple and automatable steps to derive a Normalized System from the ontological model of the B-organization, provided by applying DEMO to an enterprise, while retaining the implementation freedom of the organization under consideration. Finally we found that the impact of implementation choices is minimal and that it is clear how they affect the automated information system. With this result, one vital cornerstone for achieving enterprise agility has been covered.

Keywords: DEMO, Normalized Systems, Agile Enterprise Engineering.

1 Introduction

Agile enterprises are able to adapt rapidly and cost efficiently in response to environmental changes [1]. Examples of environmental changes enterprises have to deal with are changing market needs, changing regulations or law, and changing technologies [2]. Since most enterprise use some kind of information technology (IT) in their daily operation, the supporting information systems should be able to change accordingly. This research aims at finding key concept to link agile enterprises with agile information systems.

The Design and Engineering Methodology for Organizations (DEMO[1]) has already proven to be an effective tool in realizing agile enterprise [3]. The Normalized Systems (NS) approach has proven to be the key for developing agile

[1] http://www.demo.nl/

A. Albani, J.L.G. Dietz, and J. Verelst (Eds.): EEWC 2011, LNBIP 79, pp. 31–45, 2011.

information systems [4]. Both DEMO and NS are aimed at finding concepts for achieving agility, comprised in both a way of thinking and a way of modeling. In this paper it is explored how to link agile enterprise with agile automated information systems.

By comparing the two theories, we found that the theories adhere to the idea that separation of tasks is crucial to fully define any system, being it an enterprise or an IT system. Moreover, by looking at the modeling concepts from both DEMO and Normalized Systems, a first method for defining a Normalized System from the ontological model of the B-organization, provided by applying DEMO to an enterprise, was designed, tested and improved. Because DEMO abstracts from implementation details and our method does not fix any implementation details, the gained Normalized System still offers implementation freedom. It is shown that the impact of organizational and IT implementation choices is minimal and that it is clear how they affect the automated information system.

The remainder of this paper is structured as follows. First, some theoretical background about DEMO and Normalized Systems is outlined in section 2. Section 3 then provides a theoretical comparison between the DEMO and NS ways of thinking. In section 4, one of the cases against which the method has been tested, is introduced. The steps for deriving a model of a Normalized Systems from an ontological model of the B-organization of an enterprise are described in section 5, using the case from section 4 as an example. In section 6 we will show that there is still freedom for implementation choices, by showing how implementation choices impact the model of the Normalized System. Finally, section 7 provides the conclusions as well as directions for further research.

2 Theoretical Background

2.1 DEMO

DEMO is a methodology for the design, engineering, and implementation of organizations [3]. Applying DEMO to an enterprise provides an *enterprise ontology*: the essence of an organization, fully independent from the way in which it is realized and implemented. Dominant in the DEMO methodology is the Performance in Social Action (PSI- or ψ-) theory that underlies this notion of enterprise ontology. The PSI theory consists of four axioms and one theorem, which will be shortly explained now:

- The *Operation Axiom* states that the people in an organization (subjects) can perform two kinds of acts: *production acts* (P-acts) and *coordination acts* (C-acts). We abstract from the subjects in order to concentrate on the different *actor roles* they fulfill. An actor is a subject fulfilling an actor role. By performing P-acts they contribute to achieving the purpose or the mission of the enterprise. By performing C-acts they enter into and comply with mutual commitments regarding P-acts.

– The *Transaction Axiom* states that C-acts and P-acts occur in universal patterns, called transactions. These patterns always involve two actor roles (initiator and executor) and are aimed at achieving a particular result. The basis transaction pattern for one transaction consists of five steps, namely request, promise, execute, state and accept. In the request step the initiator of the transaction requests the production of a P-fact, in the promise step the executor of the transaction promises to produce the P-fact, in the execution step (this is the P-act) the executor produces the P-fact, in the state step the executor presents the P-fact to the initiator and in the accept step the initiator accepts the P-fact. Apart from these four types of "success" C-acts, also 7 other types of C-acts exist (Table 1).

Table 1. The 11 types of coordination acts (C-acts)

success	problems	canceling
request	quit	cancel
promise	decline	refuse
state	stop	allow
accept	reject	

– The *Composition Axiom* states that transactions (which bring about new facts in reality) are related to each other in one of two possible ways: either a transaction is enclosed in another transaction, or a transaction is self-activating.
– The *Distinction Axiom* states that there are three distinct *human abilities* playing a role in the operation of actors, called *performa, informa and forma*. The forma ability concerns the being able to handle data and documents fully ignorant of its content and meaning, e.g., to copy, transport, and store documents. The informa ability regards the intellectual capacity of human beings, the ability to reason and to compute or derive new facts from existing ones. The performa ability concerns the ability of human beings to produce original new things, i.e., facts that cannot be derived from existing facts. Examples of such facts are decisions and judgments.
– Finally, the *organization theorem* states that an enterprise is a layered integration of three homogeneous aspect systems: the B-organization (B from Business), the I-organization (I from Intellect), and the D-organization (D from Documents).

DEMO states that a complete and essential enterprise ontology is a model of the B-organization, consisting of four aspect models:

– The *Construction Model* (CM) specifies the composition, environment, and structure of the organization. It contains the identified *transaction kinds* and associated *actor roles* as well as the *information banks* and links between these banks and actor roles.

- The *Process Model* (PM) details each single transaction type of the CM according to the universal transaction pattern. In addition, the initiating and waiting relationships between transactions are shown.
- The *Action Model* (AM) specifies the imperatively formulated *business rules* that serve as guidelines for the actors in dealing with their agenda. It contains one or more action rules for each agendum type.
- The *State Model* (SM) specifies the *object classes, fact kinds, transaction result kinds*, and the declarative formulations of the *business rules*. Some of the fact kinds can be derived from original fact kinds with a so-called derivation rule.

The way of modeling will be shown by means of an example case in section 4.

2.2 Normalized Systems

Lehman's software evolution laws stipulate that 1) "A program [...] must be continually adapted else it becomes progressively less satisfactory" and 2) "As a program is evolved its complexity increases unless work is done to maintain or reduce it" [5]. On one hand this means that information systems should always change in order to be effective, while on the other hand it means that change will become more difficult over time, unless the information system is designed to be evolvable. In order to genuinely design information systems accommodating change, they should exhibit stability towards a set of anticipated changes, i.e., the impact of a change is only dependent on the nature of the change itself [4, pp.106-107]. If a change requires increasing effort as the information system grows, this is called a *combinatorial effect*. Normalized Systems are defined as information systems exhibiting stability with respect to an anticipated set of changes and avoiding the occurrence of combinatorial effects [4].

The Normalized Systems approach deduces a set of four principles, or design guidelines, to identify and circumvent combinatorial effects [4]:

Principle 1. *Separation of concerns* implies that every change driver or concern should be separated from other concerns;

Principle 2. *Data version transparency* implies that data should be communicated in version transparent ways between components;

Principle 3. *Action version transparency* implies that a component can be upgraded without impacting the calling components; and

Principle 4. *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.

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. Also, the design principles show that existing software

constructs, such as functions and classes, by themselves offer no mechanisms to accommodate anticipated changes in a stable manner. Normalized Systems therefore proposes to encapsulate software constructs in a set of five higher-level software elements, namely, *data element, action element, workflow element, trigger element* and *connector element*. These elements are modular structures that adhere to the design principles, in order to provide the required stability with respect to the anticipated changes [4]. The elements are independent of a specific technology environment. The internal structure of every element can be described by a design pattern which is executable and can be expanded automatically, enabling the automation of software development [4].

- A *data element* represents an encapsulated data construct that contains various attributes or fields with its get- and set-methods to provide access to their information in a data version transparent way. Generic supporting tasks, also called cross-cutting concerns, for instance access control and persistency, should be added to the element in separate constructs [4]. An attribute can either be a link to another data element or a primitive type such as Integer, String, Boolean, or Date. NS data elements can be represented by means of an ER-diagram[2].
- An *action element* contains a core action representing one and only one *specific functional* task - in contrast, there are *generic supporting* tasks, called cross-cutting concerns, which are not modeled in terms of action elements. Arguments and parameters need to be encapsulated as separate data elements, and cross-cutting concerns like logging and remote access should be added as separate constructs. Four different implementations of an action element are distinguished:
  - 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 fulfill the task. The user then has to set the state of the life cycle data element through a user interface, after completion of the task.
  - A process step can also require more complex behavior. A single task in a workflow can be required to take care of other aspects, which are not the concern of that particular flow. A *bridge action* creates an instance of another data element which goes through its own designated flow. At this point it is assumed that any data element instance that is created by another instance reports its status to the instance it is created by. In that way it is possible for the creating instance to wait for some state of its 'child(ren)' to continue.
  - When an existing, external application is already in use to perform a certain action, the action element can be implemented as an *external action*. Such an external action calls another (information) system and sets its end state depending on the external systems' reported answer.
- Although workflows can be considered as a derivative concept from actions, based upon the first and fourth principle, a workflow has to be separated

[2] For details see, e.g., http://en.wikipedia.org/wiki/Entity-relationship_model

from other action elements [4, p.119]. A *workflow element* contains a sequence of actions, which should be executed in order to fulfill a workflow, and intermediate states, and can be described by a state transition diagram. A consequence of the state full workflow element 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. This data element is called the life cycle data element of a flow. It is considered every data element knows a workflow.
- The *trigger element* is for controlling the states (both regular and error states) and checking whether an action element has to be triggered. For every action element a trigger element can be fully and deterministically derived.
- Finally, the *connector element* enables users as well as other applications from external systems to interact with data elements while ensuring an action element is not called in a stateless way. For every data element a connector element can be fully and deterministically derived.

Both data and action elements can contain cross-cutting concerns, or *generic supporting tasks*. So, next to identifying the elements, the supporting tasks have to be identified. The required specifications to define a Normalized System are summarized in Table 2. As said earlier, the trigger and connector elements need not be specified.

Table 2. Required specifications for modeling a Normalized System

Required specification		**Format**
Elements	Data elements	Name, set of attributes (name, type)
	Action elements	Name, type, description/pseudo code
	Workflow elements	Name, state transition diagram
Cross-cutting concerns		Name, description

3 Theoretical Comparison

When comparing the theories, for DEMO, one only has to focus on the four axioms of the PSI theory; the organization theorem of the PSI theory is deduced from the four axioms. For Normalized Systems it holds that, while the latter three principles are completely taken care of by the elements, it is up to the designer to fully adhere to the (first) principle of separation of concerns[3]: "the more fine-grained the identification of the tasks by a designer, the more tasks are separated from each other" [4, p.112]. Normalized Systems identifies two different kinds of tasks: functional tasks (action elements) and supporting tasks (cross-cutting concerns).

[3] One could even say that NS-principles 2, 3 and 4 are special cases of Separation of Concerns (Principle 1).

DEMO clearly helps in identifying the different functional tasks. The operation axiom discerns two different types of tasks, namely, 1) tasks regarding coordination, and 2) tasks regarding production. The transaction axiom identifies the different transaction steps, which are the same for every transaction. Finally, the distinction axiom defines tasks at a more fine-grained level while also discerning those as being from different kinds, namely, ontological, infological or datalogical. The question now rises how DEMO helps in identifying the supporting tasks.

For creating information banks, cross-cuttings concern persistency for data elements and logging for action elements can be defined. Creating these banks are in fact datalogical acts. So, it might be concluded that the I- and D-organization in fact prescribes the generic supporting tasks. Another cross-cutting concern, which is present in many organizations, is authorization. However, as an organization can choose whom to authorize and in what way, authorization is considered part of the implementation of an organization and therefore it is not present in an ontological model. In another research, we investigate how that part, in fact a cross-cutting concern for all organizations, can be modeled.

4 Enterprise Ontology of the Dutch Governmental Subsidy Schemes

A subsidy is an entitlement to financial resources provided by an administrative authority for the purposes of specifically named activities of the applicant, other than by way of payment for goods or services supplied to the administrative authority [6, art.4:21.1]. In the Netherlands, for different kinds of activities, different subsidy schemes exist. The process of granting a subsidy is the same for all individualized (as opposed to portfolio) schemes (Figure 1), which simplified reads as follows: the applicant (CA01) applies for a subsidy (T01) at the subsidy granter (A01). Before the subsidy can be granted, a formal check has to be performed (T02). Next, the application will be substantively examined, yielding an amount the applicant will receive for his activities (T03). If it turns out the application does not fulfill the substantive requirements, the amount will be 0 (zero). Only if the formal evaluation is passed and a non-zero amount is determined for the application, the subsidy will be granted. Both the formal evaluator (A02) and amount determiner (A03) need access to the particular subsidy scheme act for knowing the formal criteria and the rules for amount determination. Dutch law states that payment (T04) has to follow within four weeks after the grant. Waiting conditions are indicated by a dashed line in the Process Model (Figure 2). The wait condition from T02/ac to T01/dc means the subsidy grant can be declined early in the process, namely when the formal evaluation yielded a negative result. The promise can only be given when the amount determination has been finished (for details, see the Action Model in Listing 1).

From the State Space Diagram (Figure 3) it can be read a subsidy has a derivation rule for whether it should pass the formal evaluation or not, and

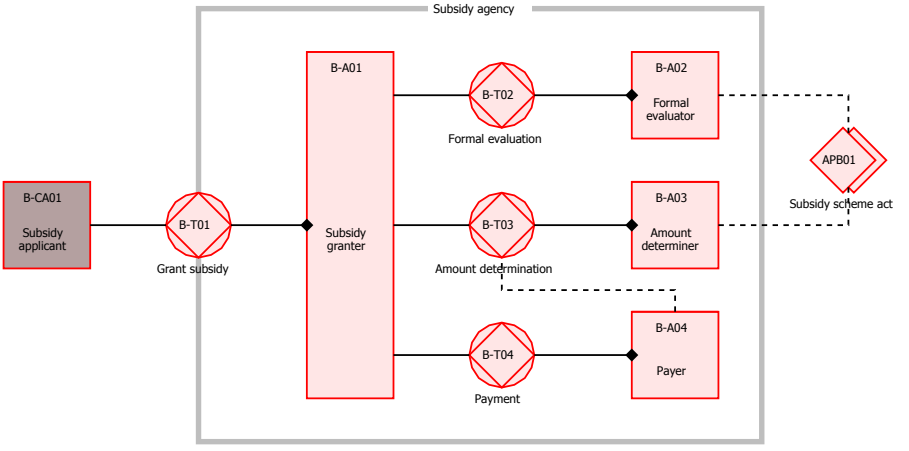


Fig. 1. Organization Construction Diagram (part of CM) of a subsidy agency

Table 3. Transaction Result Table (part of CM) of a subsidy agency

T01 Grant subsidy	R01 [subsidy] has been granted
T02 Formal evaluation	R02 [subsidy] has been formally evaluated
T03 Amount determination	R03 amount for [subsidy] has been determined
T04 Payment	R04 amount for [subsidy] has been paid out

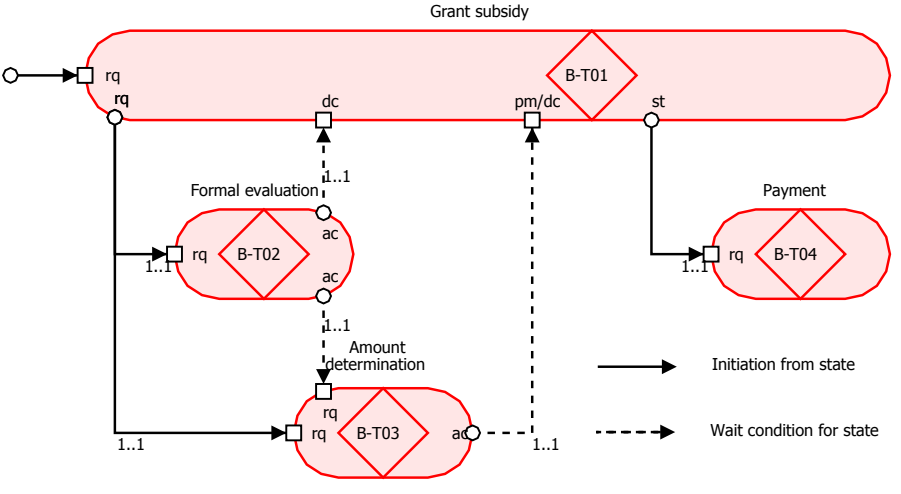


Fig. 2. Process Structure Diagram (part of PM) of a subsidy agency

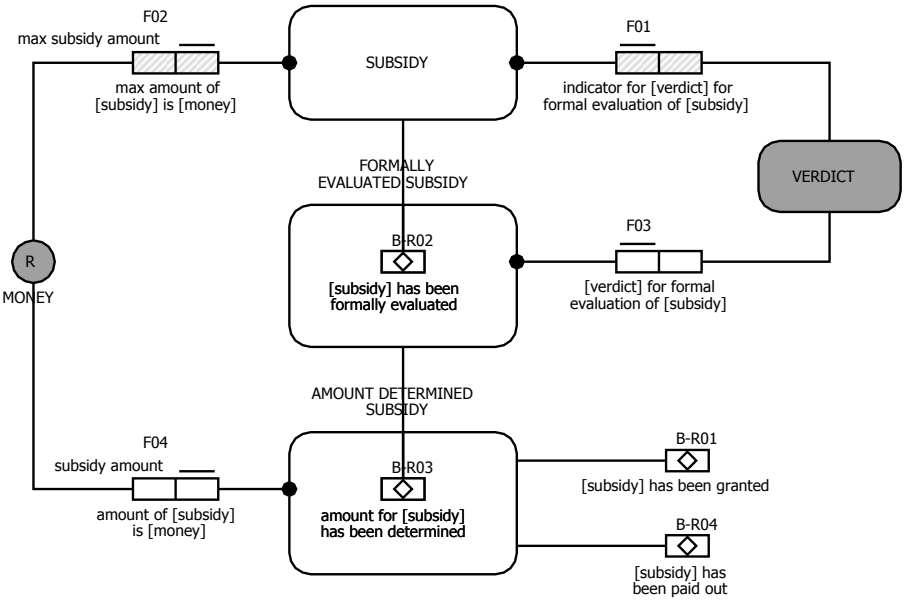


Fig. 3. State Space Diagram (part of SM) of a subsidy agency
(Notation is like ORM: http://orm.net/)

one for calculating the maximum subsidy amount. The definitions of the rules in general differ per subsidy scheme. Also in the SM, some process steps are shown: First, a subsidy is evaluated formally, using the derived fact kind for the verdict indicator, producing a result for the formal evaluation. A verdict in general is either positive ("OK") or negative ("not OK"). Second, for a subsidy the subsidy amount is determined, using the derivation rule for the maximum amount, producing the amount of subsidy that is determined. Only then, the subsidy can be granted and paid out.

The Action Models states for every actor role how to deal with an event type. The action rules for actor role A01 are provided in Listing 1. The action rules for the other actor roles are straightforward as they don't contain any decision rules. The action rules are in fact guidelines for the subjects; it is possible for an actor to deviate from the guidelines.

With regard to enterprise agility aspects, within the automated information system supporting the subsidy grant process, it must be possible to:

1. add rules for different subsidy schemes;
2. change rules following from a change in law for either the entire process or a specific scheme;
3. add different communication channels (following from the Dutch Reference Architecture NORA);
4. choose to automate tasks or perform them manually;
5. switch between internal and external data usage;

```
 1  when T01 is requested
 2  then T02 must be requested
 3
 4  when T02 is stated
 5  then T02 must be accepted
 6
 7  when T02 is accepted
 8  if T02 yielded a positive verdict
 9  then T03 must be requested
10  else T01 must be declined
11
12  when T03 is stated
13  then T03 must be accepted
14
15  when T03 is accepted
16  if   subsidy amount > 0
17  then T01 must be promised
18  else T01 must be declined
19
20  when T01 is promised
21  then T01 must be executed
22       T01 must be stated
23
24  when T01 is stated
25  then T04 must be requested
26
27  when T04 is stated
28  then T04 must be accepted
```

Listing 1. Action rules for A01 of subsidy agency

6. choose which information is shown on a screen;
7. outsource or insource parts of the system;
8. assign tasks and task types to people; and
9. add controls for managing throughput time.

5 Deriving a Normalized System from an Enterprise Ontology

Step 1. *State Model*

For each object class, scale, or category, a data element is created. For the subsidy case, five data elements are created: Subsidy, FormallyEvaluatedSubsidy, AmountDeterminedSubsidy, Money, and Verdict. External object classes, scales, and categories (shaded gray) must be represented by one or more primitive types. For the subsidy case, Money is represented by a floating point number, and Verdict is represented by a String. For Verdict, we could also have chosen a Boolean, however, the Normalized Systems approach prescribes Booleans should be avoided as they lock up the possibility to have more than two values; Booleans should only be used in rare cases. Fact types, connected to object classes with reference laws, are represented by links between the data elements: FormallyEvaluatedSubsidy contains a link to Subsidy and a link to Verdict, and AmountDeterminedSubsidy contains a link to FormallyEvaluatedSubsidy and a link to Money (Figure 4).

For the unicity law, dependency law, and exclusion law from DEMO, it is not immediately clear how they are supported in Normalized Systems. Additional actions or other means of constraints are needed. Moreover, derived fact types cannot simply be modeled in a Normalized System. A derivation rule could be implemented in an action element, but then still it is not clear how and when to use it: the calculation should not be performed more often than strictly necessary, however, its result is not valid anymore when the original facts on which it is based, change.

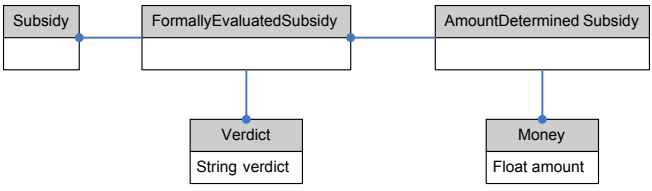


Fig. 4. ER diagram of identified NS data elements for subsidy agency

Step 2. *Construction Model*

For each transaction in the CM, two data elements are created: one for the initiator and one for the executor, both knowing their own workflow element with action elements as exhibited in Figure 5. If the initiator or the executor of a transaction relies only outside the organization, the data element, workflow, and action elements for that part can be left out. The dotted arrows indicate the following. In the request action of the initiator, the data element for the executor is created - therefore, the communication link is going from an action to a state, being the start state of the flow of the newly created data element. In the promise and state action of the executor, the initiator is informed. In the accept action of the initiator, the executor is informed. To accommodate the being informed, wait actions are introduced in the other flow. Since the initiator does not know whether the executor is going to promise or decline, it will wait for either message. A similar reasoning holds for the accept/reject of the initiator. In the basic flow however, only the success paths are fully outlined. The workflows can be extended in order to accommodate all problem courses, i.e., decline, reject, quit, and stop. Huysmans et al. already showed how to deal with the cancelation patterns [7].

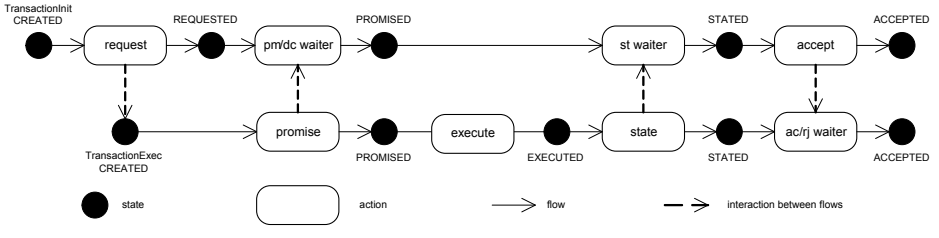


Fig. 5. Basic NS workflow patterns for a single DEMO transaction kind

For the subsidy case, seven transaction data elements are created: two for each of the three internal transactions (T02, T03, and T04) plus one for the customer-initiated transaction (T01). Every data element representing part of the transaction pattern contains links to the data elements that were created for the object classes the transaction result is about. Since every transaction is about a subsidy, all data elements contain a link to the Subsidy

data element. Actor roles are fulfilled by subjects in an organization. A cross-cutting concern authorization should provide that a person only can access the data and perform the actions he or she is authorized for. Regarding access to fact banks, for each information link one additional connector element should be defined that make sure that an actor role can see the data he needs for his work.

Step 3. *Action Model*

From the Action Model, the decision rules can be read as well as the exact order in which different transactions are started (workflow). Additional actions must be introduced if another transaction must be started (bridge action) or when to wait for some state of another transaction. Van Nuffel et al. already found that the decision rule should be placed in another action, as the decision can change independent from the way in which the following action is performed and how to deal with the cancelation patterns [8]. Although not shown in the example, it was also found that the handling of multiple instantiations of the same transaction kind, e.g., for periodic payment, must be split off into yet another flow. The number of initiated transactions can be read from the Process Model.

For the subsidy case, additional identified actions are: T02creator, T02waiter, T02verdictchecker, T03creator, T03waiter, T03amountchecker, and T04creator, following from respectively lines 2, 7, 8, 9, 16, 17, and 26 of Listing 1. The complete workflow for T01exec is shown in Figure 6, the other flows don't deviate from the basic workflow as shown in Figure 5.

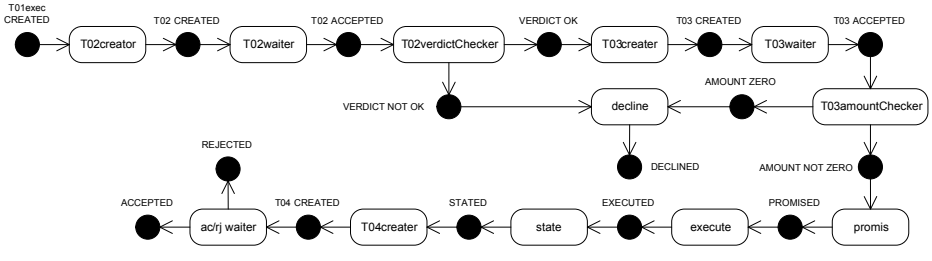


Fig. 6. Workflow for the T01exec data element

6 Implementation Freedom

After applying the steps above, one gets a Normalized System that supports the operation of an enterprise. Now we will show that the desired enterprise agility is achieved.

1. Add rules for different subsidy schemes: Because of data version transparency, different implementations of the subsidy data element for different schemes can be introduced. For example, for one scheme, the subsidy is about cars, while for another scheme, it is about houses. Because of action version transparency, different actions that apply for the different kinds of

subsidy can be implemented as well. For different schemes, the action for calculating the maximum amount will differ. Of course, there must be some mechanism that ensures the data and action versions are matched.
2. Change rules for a specific scheme: Same as for adding a new scheme.
3. Change rules for the entire process: Depending on the nature of the change, this will mean implementing a new version of some action or adding or deleting some action.
4. Add different communication channels: Because of action version transparency, different versions of a 'coordination action' can be implemented. In general, to enable re-use, 'coordination actions' will be put in their own designated flow, which will be started by means of a bridge action.
5. Automate tasks or not: Again, this is up to the implementation of the action. If it is chosen to automate the task, some code must be provided. When automated, again, different versions can be used.
6. Switch between internal and external data usage: For example, for getting person data, e.g., about the applicant of a subsidy (currently not modeled), the organization could use its own database in which it must put all person data on first use. However, one could also choose to connect to the governmental persons register. Because of data version transparency, it is easy to change. If external data is used, personal data can be seen as derived facts.
7. Choose which information is shown on the screen: A screen is user interaction which is provided by connector elements. Creating a screen means adding a connector element.
8. Outsource (or insource) part of the system: A similar reasoning holds as for changing rules for the entire process: either actions are deleted or they are re-implemented for calling an external system. Interesting is that this agility in information systems does not only improve the agility in organizations in splitting, but also eases the allying with new partners [9, p.102].
9. Add controls for managing throughput time: This can be reached by adding trigger elements with timers. For example, if the payment of some application is not performed within three weeks after granting, some notify action must be started.

7 Conclusions and Future Research

In linking DEMO and Normalized Systems, we found that the underlying theories both prescribe a modular structure and share the idea that separation of tasks is crucial to fully define a system. We also found that an ontological model of the B-organization of an enterprise, provided by applying DEMO to an enterprise, suffices to create a functionally complete automated system for supporting the operation of the enterprise while also providing implementation freedom and thus enterprise agility support. This freedom includes, but is not limited to, using different information channels, choosing to automate tasks or not, and changing business rules. The system is however not designed taking into account aspects such as user-friendliness, including graphical user interfaces, and

Table 4. Mapping DEMO concepts onto NS primitives and v.v.

DEMO concepts / NS primitives	Data element	Action element	Workflow element	Trigger element	Connector element	Cross-cutting concern
Object class, category, scale dimension	X				?	
Fact kind	X				?	
Existence laws	(X)					
Derived fact kind						
Actor role						X
Transaction kind	X				?	
Transaction step		X		?		
Information bank	X				?	X
Information link					?	
Action rule		X	X	?		

other non-functional requirements, e.g., performance, as these aspects are not present in DEMO. The Normalized Systems approach considers these aspects as cross-cutting concerns which should be addressed separately from the functional requirements.

We also found that Normalized Systems does not yet support all concepts of DEMO. In Table 4 we have summarized for each DEMO concept which NS primitives have to be created, indicated by an X. One empty row appears: for derived fact kinds it is not yet clear how to model them in a Normalized System; there is not a construct in NS onto which it can directly be mapped. The (X) in the third row means that the existence laws are only partially covered by Normalized Systems. It should be investigated how one can model all the existence laws from DEMO in a Normalized Systems while retaining agility.

The other way around, from Table 4 it can also be read for each NS primitive which DEMO concepts are needed for defining it. There are two empty columns (except for the question marks), for the trigger and connector element as they can be derived from the other elements.

With this result, one vital cornerstone for achieving enterprise agility has been covered. At the same time, in achieving enterprise agility, more means and concepts are needed.

In this research we found some issues that need further investigation:

- First, it must be investigated how to model derived fact kinds, derived object classes, and derived categories in Normalized Systems; there is not one construct in Normalized Systems present that supports the derivation of facts.

- Second, for the unicity law, dependency law, and exclusion law from DEMO, it is not immediately clear how they are supported in Normalized Systems, so it should be investigated how Normalized Systems can support these laws.
- Third, as indicated by the question marks in Table 4, this research did not fully look into the trigger and connector element.
- Fourth, it should be investigated to what extent modeling the I- and D-organization of an enterprise - based on the DEMO models for the B-organization - adds value for creating automated systems according to the Normalized Systems approach. De Jong [10] describes how to derive DEMO models for the I- and D-organizations from the DEMO models from the B-organization. During our research we noticed that the I- and D-organization seem to offer mainly generic functionality, or, in Normalized Systems terminology, cross-cutting concerns. Therefore it should be investigated to what extent the I- and D-organization offer *generic* functionality and to what extent *specific* functionality.

References

1. Tsourveloudis, N.C., Valavanis, K.P.: On the Measurement of Enterprise Agility. Journal of Intelligent and Robotic Systems (33), 329–342 (2002)
2. Arnold, B., Engels, A., Op 't Land, M.: FPS: een andere kijk op componenten en architectuur in de financiële wereld (deel 2). A & I, 24–32 (2000)
3. Dietz, J.L.G.: Enterprise Ontology. Springer, Heidelberg (2006)
4. Mannaert, H., Verelst, J.: Normalized systems: re-creating information technology based on laws for software evolvability. Koppa, Kermt (2009)
5. Lehman, M.: On understanding laws, evolution, and conservation in the large-program life cycle. Journal of Systems and Software 1, 213–221 (1980)
6. De Rijksoverheid : (Algemene wet bestuursrecht), http://www.rijksoverheid.nl/onderwerpen/algemene-wet-bestuursrecht-awb
7. Huysmans, P., Bellens, D., Van Nuffel, D., Ven, K.: Aligning the Constructs of Enterprise Ontology and Normalized Systems. In: Albani, A., Dietz, J.L.G. (eds.) CIAO! 2010. Lecture Notes in Business Information Processing, vol. 49, pp. 1–15. Springer, Heidelberg (2010)
8. Nuffel, D.V., Huysmans, P., Bellens, D., Ven, K.: Translating Ontological Business Transactions into Evolvable Information Systems. In: 5th International Conference on Software Engineering Advances, ICSEA 2010 (2010)
9. Op 't Land, M.: Applying architecture and ontology to the splitting and allying of enterprises. PhD thesis. Delft University of Technology (2008)
10. de Jong, J., Dietz, J.L.: Understanding the realization of organizations. In: Albani, A., Dietz, J.L. (eds.) CIAO! 2010. Lecture Notes in Business Information Processing, vol. 49, pp. 31–49. Springer, Heidelberg (2010)

Transformation of DEMO Metamodel into XML Schema

Yan Wang[1], Antonia Albani[2], and Joseph Barjis[3]

[1] Tilburg University
European Research Institute in Service Science
PO Box 90153, 5000 LE Tilburg, The Netherlands
y.wang13@uvt.nl
[2] University of St. Gallen
Institute of Information Management
Müller-Friedberg-Strasse 8, 9000 St. Gallen, Switzerland
antonia.albani@unisg.ch
[3] Delft University of Technology
Faculty of Technology, Policy and Management
PO Box 5031, 2600 GA Delft, The Netherlands
j.barjis@tudelft.nl

Abstract. In this paper, we propose an approach to transform models derived by applying the Design and Engineering Methodology for Organizations (DEMO) into an exchangeable format. DEMO is based on a founded theory, the Ψ-theory, and satisfies the requirements to be a well defined domain modeling methodology. Having the DEMO models represented in an exchangeable format is beneficial for different types of applications supporting the information system development process. Applications used for the automatic analysis (simulation) of the DEMO models or for the identification of business components are just two examples to be mentioned.

Keywords: DEMO metamodel, model transformation, XML Schema.

1 Introduction

Modern enterprises are challenged by the reality of a dynamic business environment. With the increasing business scale, companies are encountering more complex situations, such as globalized sales and sourcing markets, shortened product life cycles, and innovative pressure on products, services and processes [1]. In order to stay in and win the game in the competitive business world, enterprises need to adapt to the fast changing market quickly and expand their cooperative relationships with their business partners. Adaptive and agile information systems for enterprises are therefore crucial in order to support the business needs. Thus while developing such information systems for enterprises, it is necessary to have a suitable methodology for modeling the business domain. The appropriateness and the quality of the business domain models are vital for the

A. Albani, J.L.G. Dietz, and J. Verelst (Eds.): EEWC 2011, LNBIP 79, pp. 46–60, 2011.

development process. Dietz proposes some quality criteria regarding a business domain model in [2], which are *coherence* (the domain models constitute a logical and truly integral whole), *comprehensiveness* (complete coverage of all relevant issues), *consistency* (the domain models are free of contradictions or irregularities), *conciseness* (all relevant models are compact and succinct), and *essence* (the domain model should only show the essence of the enterprise). The *Design and Engineering Methodology for Organizations (DEMO)* [2] is a methodology that satisfies these requirements and that additionally distinguishes between essential (business), infological and datalogical acts and facts, ultimately resulting in the distinction between three aspect organizations: the B-organization (B from business), the I-organization (I from infological), and the D-organization (D from datalogical). The apparent advantages of DEMO – in particular the huge reduction of complexity [3] – led us to choose DEMO for modeling the business domain.

DEMO has already proven to be beneficial for the development of the supporting information systems (see [4,5]). However no exchange format of DEMO models has been defined so far in order to allow for automatic access of DEMO model data by third party applications. Example applications are a) tools for the automatic identification of business components based on DEMO models, or b) simulation tools, where readability and structure of the model data are both important and helpful. This paper therefore presents research on metamodel transformation, where the transformation of DEMO metamodel to XML Schema is elaborated.

The paper is structured as follows. Section 2 introduces state of the art in model transformation. Additionally, the section presents the metamodel transformation approach as used in this paper. The DEMO metamodel which is needed for the said transformation is introduced in the sections 3. In section 4 several transformation rules that are used to transform the DEMO metamodel into XML Schema are introduced. One exemplary case, in section 5, shows the transformation procedure and the resulting XML structure for an instance of a DEMO metamodel. The evaluation of the transformation results is made in section 6. At last, section 7 concludes and proposes some work for future research.

2 Model Transformation

Metamodeling has been widely used as a transformation approach, of which exemplary applications can be found in [6] [7]. Figure 1 visualizes a general metamodel mapping transformation. The transformation is from Model A to Model B. The language used for expressing Model A is defined in Metamodel A. The Metamodel B defines the language that describes Model B. These two languages are connected by the Transformation model. The task of this transformation model is to map the source language and the target language, so that it can build a bridge from Model A to Model B. The transformation model is expressed in terms of a set of transformation rules that address the detailed guidance of the transformation. Thus the transformation procedure from Model A to Model B is accomplished by using metamodeling.

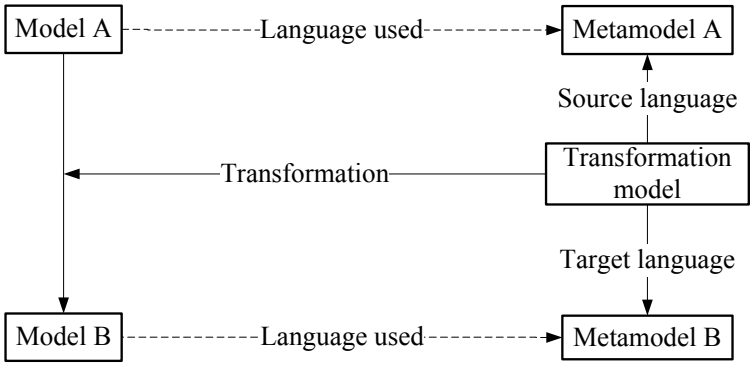


Fig. 1. Metamodel Mapping Transformation [8]

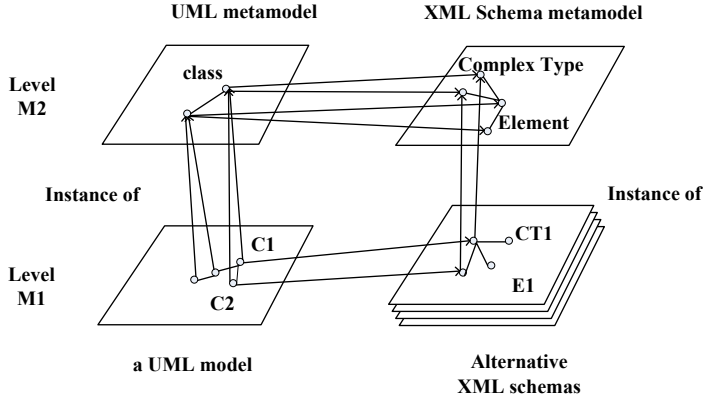


Fig. 2. UML to XML Schema Transformation [9]

A good example of applying metamodeling e.g., in the Model Driven Architecture (MDA), is the transformation between UML class diagram and XML Schema from [9]. Depicted in Figure 2, the source UML model and the target alternative XML Schema are situated in level M1, the source UML metamodel and the target XML Schema metamodel are situated in level M2. The source and target models are mapped with each other within the same level. For example, the class construct from UML metamodel, which is in level M2, is possible to be mapped either to "Element" declaration or "Complex Type" definition construct from XML Schema metamodel, which is also in level M2. Following the same transformation rules, instances as e.g., "C1" or "C2" of the UML metamodel "class" type can then be mapped either to instances of the "Element" or "Complex Type" of the XML Schema metamodel.

When using the metamodeling approach in model transformation, the actual transformation is executed within the modeling languages used in the metamodel

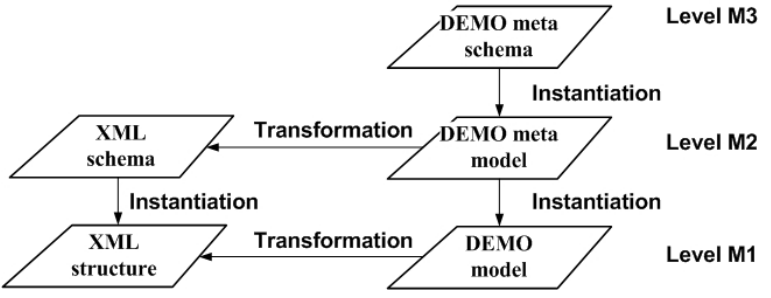


Fig. 3. DEMO metamodel to XML Schema transformation

of the source and target models. The complex instance contents at the model level are left out of sight during the transformation. In the presented research we use the DEMO metamodel defined by Dietz and introduced in [10] as the source, and the XML Schema as the target metamodel on the level M2. Fig. 3 illustrates the elements used for the transformation needed. The DEMO meta schema on the level M3 defines the concepts used for defining the DEMO metamodel. Based on the DEMO metamodel and the XML Schema on level M2, transformation rules are defined allowing the transformation of DEMO models into XML format on level M1.

3 DEMO Metamodel

As metamodeling transformation approach is chosen in this proposed DEMO transformation, DEMO metamodel is a crucial source material for the transformation. The DEMO metamodel [10] on level M2 (see Fig. 3), corresponding to the aspect models on level M1, includes the Meta Construction Model (MCM), Meta Process Model (MPM), Meta Action Model (MAM), Meta State Model (MSM), and the metamodel for cross-model tables TRT, IUT and BCT. These metamodels provide the basic constructs and relationships that occur in any instance aspect of DEMO models on level M1 (see fig. 3). The language used in specifying the DEMO metamodel is called World Ontology Specification Language (WOSL). Regarding the limited space in this paper, we restrict ourselves on describing two aspects of the DEMO metamodel, namely the MSM and the MCM. For the same reason, only these two are discussed in the following sections. We refer to [10] for readers who are interested in a complete description of DEMO metamodel.

3.1 Meta State Model (MSM)

We start by introducing the MSM, since it builds the basis for the whole DEMO metamodel. The MSM is part of the DEMO metamodel, and has connection with

the MCM. It fulfills two roles, besides being the metamodel of SM, MSM also defines the metamodel of the whole DEMO metamodel. We call the metamodel of the whole DEMO metamodel a *meta schema* (see level M3 in Fig. 3). We are going to briefly describe the MSM's role of being the meta schema and focuse on its specification of being metamodel of SM, because the said transformation takes place on level M2 (Fig. 3).

The MSM, as DEMO meta schema, specifies the concepts used in WOSL, including the declared fact type, derived fact type, unary fact type, binary fact type, object class, scale and category. Furthermore, existence laws, such as exclusion law, unicity law, reference law and dependency law are defined in this meta schema as well.

The MSM, as being the metamodel of SM, defines the basic constructs of the SM. A SM is instantiated in terms of a lawful set of basic constructs. Figure 4 shows how the MSM specifies a basic construct, and how a basic construct in SM look like. The basic construct in SM (Figure 4 (b)) states that the role x in fact type F has a domain A. And for every a ∈ A there must be a tuple <a>in F. The corresponding definition of this construct in MSM is expressed in Figure 4 (a): for every unary fact type there must be one and only one instance fact type x in the binary fact type <x, y>, the domain of fact type x is the object class y. Additionally, object x is dependent on object y, while x belongs to class UNARY FACT TYPE, and y belongs to an extension of unary fact type.

The two roles of MSM imply that the MSM not only defines the structure of instance SM, but also defines the concepts used by the whole DEMO metamodel. The MCM in the following section will be explained by using the concepts defined in the MSM.

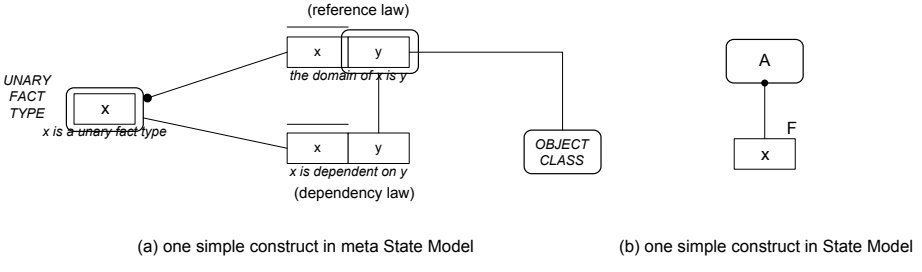


Fig. 4. Basic construct at both metamodel level and model level (with dependency law)

3.2 Meta Construction Model (MCM)

The MCM specifies the *interaction structure* of CM. Being outlined in Figure 5, TRANSACTION KIND, ELEMENTARY ACTOR ROLE and INFORMATION BANK including the PRODUCTION BANK and COORDINATION BANK are the core object classes within the MCM.

For describing the DEMO metamodel we adopt the following convention (we take the type ELEMENTARY ACTOR ROLE as an example): if there is a fact

a, and the type of a is the elementary actor role, we would say that 'a is an elementary actor role' or 'elementary actor role a'. This is equivalent saying that 'elementary actor role (a) holds'. This kind of expression will be used in the specification of DEMO metamodel in this section.

The description of the correlation between the ELEMENTARY ACTOR ROLE and TRANSACTION KIND is specified like this: if there is an 'elementary actor role a is an initiator of transaction kind t', there must be a 'transaction kind (t) holds'. Meanwhile, constrained by the dependency law, for every transaction kind, there must be an elementary actor role a, that 'a is an initiator of t holds', where t is the transaction kind. If there is an elementary actor role 'a is the executor of transaction kind t', there must be a 'transaction kind (t) holds'. Meanwhile, constrained by the dependency law, for every transaction kind, there must be an elementary actor role a, that 'a is the executor of t holds', where t is the transaction kind. Note that there are unicity laws hold for both a and t in the lawful binary fact type. That means every transaction kind and every elementary actor role cannot occur more than once in the lawful binary fact type. Thus it implies that there is a strict one-to-one relationship between the transaction and its executor.

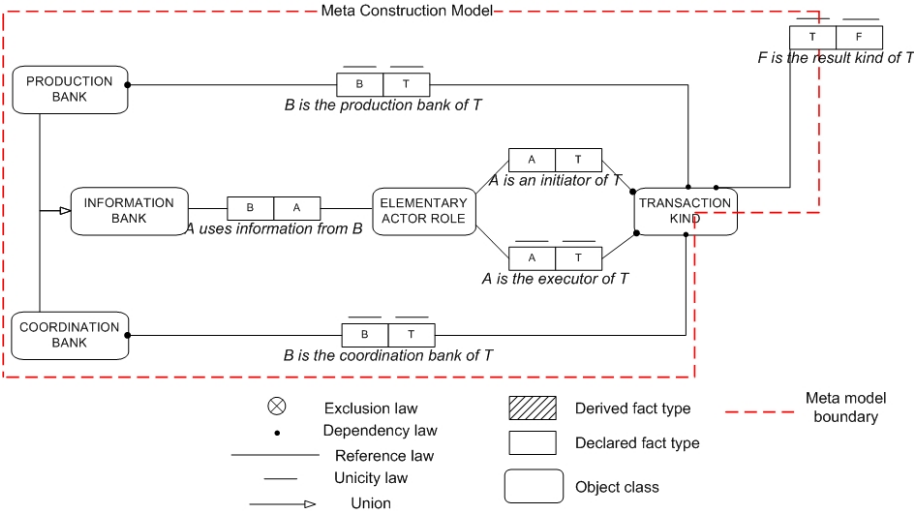


Fig. 5. Meta Construction Model

The MCM also specifies that an elementary actor role, either initiator or executor, may use information from information banks, but not necessarily. There are two kinds of information banks, namely, the production bank and coordination bank. The correlation between transaction kinds and information banks are restricted to a one-to-one relationship, as shown in Fig. 5 there are unicity laws hold in the binary fact types between transaction kinds and information banks respectively. In addition, constrained by the dependency laws hold for transaction

kinds and information banks, every coordination fact or production fact produced by a transaction must belong to the corresponding coordination bank or production bank respectively; every coordination bank or production bank can only come into existence when there is some coordination fact or production fact respectively. It is not possible to have an empty information bank or a transaction without its corresponding information banks, since there are always coordination facts and production facts produced as the results of the coordination acts and production acts which are performed during transaction process steps.

The result types are the connections between the MCM and the MSM. The transaction kind plays a crucial role in connecting MCM to MSM. Seen in Fig. 5, in the binary fact type 'F is the result kind of T', transaction kind T belongs to TRANSACTION KIND which is part of MCM. Result kind F, situates outside the MCM boundary, belongs to declared fact type which is part of MSM. The connection between the transaction kind and its result kind is strictly one-to-one relationship, as marked by the unicity laws.

An exemplary instance CM of a library case include an Organization Construction Diagram (OCD) (Figure 6) and a TRT (Table 1). In the exemplary transaction 'membership registration' (T01), the initiator of T01 is the composite actor role 'aspirant member' (CA02); the executor of T01 is the elementary actor role 'registrar' (A01). The composite actor role could be an elementary actor role or a composite actor role. The transaction symbol has two meanings in OCD, one is the transaction T01, the other one is the combination of production bank PB01 and coordination bank CB01 that belong to transaction T01. The elementary actor role A01 uses information from the composite production bank 'personal data' (CPB11) and 'general data' (CPB14). For T01, the result type is R01 'membership M has been started'. This structure is specified in MCM,which implies that the relationship between actor roles and transaction in the instance model is fully consistent with the corresponding structure in the metamodel.

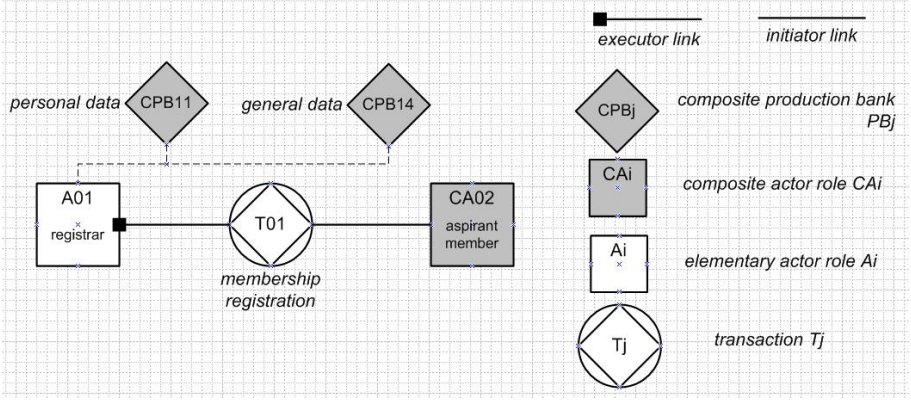


Fig. 6. An exemplary OCD of the library

Table 1. The TRT of the library

transaction type	result type
T01 membership registration	R01 membership M has been started

4 Transformation Rules

The transformation rules will be derived from three aspects: what information needs to be transformed (selection rules); how the selected information is structured in target format (structuring rules); how to verify the transformation results (mapping rules).

4.1 Selection Rules

Selection rules aim to clarify the complete and essential information objects for the transformation.

In this paper, we select information from MCM and MSM. For MCM, we select the object classes, binary fact types and the correlation among them. It is because the object classes contain all the information objects, and the binary fact types and the correlations contain the relationships among the information objects.

Considering the fact that the proposed transformation is about DEMO metamodel on level M2 (Fig. 3), we only pick up the object classes and fact types that are used in the basic constructs, and the existence laws which are directly used in building the DEMO metamodel, but exclude the concepts for specifying object class, fact types and existence laws are situated on level M3, namely DEMO meta schema. Tables 2 lists all the information objects chosen from MCM and MSM. The selected information will be transformed into target XML-based format.

Table 2. The selected information from the MCM and MSM

	Information Items		Information Items
MCM	TRANSACTION KIND	**MSM**	FACT TYPE
	A is an initiator of T		OBJECT CLASS
	A is the executor of T		the domain of x is y
	ELEMENTARY ACTOR ROLE		x is a declared fact type
	A uses information from B		SCALE
	INFORMATION BANK		PROPERTY
	PRODUCTION BANK		c is the domain of x
	B is the production bank of T		s is the range of x
	COORDINATION BANK		mutual exclusion holds for x and y
	B is the coordination bank of T		x is dependent on y
	F is the result kind of T		unicity holds for x

4.2 Structuring Rules

The chosen contents from the DEMO metamodel must be structured hierarchically in the XML schema files. It implies that those information objects must be clustered as elements, complex types or attributes in XML Schema, regarding their different features and priorities in DEMO metamodel. An example of the general structure of XML document is shown in Listing 1.1.

Listing 1.1. An example of defining the element and complex type and attribute

```
<xsd:element name="ELEMENT" type="COMPLEX_TYPE"/>
<xsd:complexType name="COMPLEX_TYPE">
    ...
<xsd:attribute name="attribute" type="type" use="required/optional"/>
</xsd:complexType>
```

We define those which have the central position in the metamodel as the root element in the schema, e.g. the one which holds the most dependency laws, or the one which is the most generically constructed in the metamodel. For instance, the transaction kind has the central position in MCM, which is because in MCM the transaction kind holds the most dependency laws. Its central position also makes sense at the model level, since in DEMO aspect models, CM is the most concise model as mentioned in section 3.2, while the other models detail part of the CM. The other information, which is not in the central position, is defined as complex types in order to detail the root element. The existential constraints contained in the metamodels are defined as attributes of either elements or complex types.

Besides considering the hierarchy in the chosen contents, we also pay attention to the structure of the instance XML files. We need to add an additional root element to have the structured model information nested in the root element, when the current root element is instantiated by more than one cases.

Listing 1.2. XML schema for global element <Transactions>

```
<xsd:element name="Transactions">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Transaction" type="TRANSACTION_KIND" maxOccurs="
       unbounded"/>
   <xsd:element name="Transaction" type="TRANSACTION_KIND" maxOccurs="
       unbounded"/>
   <xsd:element name="Transaction" type="TRANSACTION_KIND" maxOccurs="
       unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

An example of the root element in the schema of the CM explains itself (Listing 1.2). The child element <Transaction>and its complex type content "TRANSACTION_KIND" comprehends all the model information within the transaction. If there are three transactions T01, T02 and T03, for each transaction, it is specified as an element <Transaction>and includes its complext type "TRANSACTION_KIND". Then we need to set another parent element for this sequence of <Transaction>. Thus we set element <Transactions>as the root element, which includes all the <Transaction>elements in the instance XML

file. This structuring rules are applied to transform the model information from source format to target format.

4.3 Mapping Rules

This rule is made to guarantee the information completeness and correctness in the transformation results, in order to verify the transformation results. The mapping is made from two perspectives, which are the precision of the constraints in the XML schema files and the completeness of the information objects in the instance XML documents, compared with the original DEMO metamodel. Therefore, for each information object in the original DEMO metamodel, there should be a corresponding interpretation in the XML schema files, in terms of either an element or a complex type; for every necessary constraint, there should also be an equivalent part in the schema files, in terms of an attribute of either an element or a complex type.

The following table lists how the proposed transformation converts DEMO meta schema elements in XML Schema elements at a high level. The DEMO meta schema elements are the elements specified in WOSL, including the object class, the fact type, the exclusion law, the unicity law, the reference law and the dependency law. The object class is converted to either an element or a complex type content. The fact type is considered as either a child element or a contained element within complex types. The exclusion law restricts the value of an XSD string simple type. The unicity law notifies the occurrence of a contained element within complex types. The reference law and the dependency law tell the necessity of the usage of an attribute.

Table 3. DEMO meta schema elements and XML Schema elements

DEMO meta schema elements	**XML Schema elements**
Object class	Element Complex type
Fact type	Child element of a contained element (complex type)
Exclusion law	Restriction of an XSD string simple type with two enumeration facets
Unicity law	Occurrence indicators of elements •minOccurs = 1 •maxOccurs = 1
Reference law	Attribute of an element •use = optional
Dependency law	Attribute of an element •use = required

5 Exemplary Case

In this section, we take one transaction of the library case as an example to demonstrate the transformation procedure. As exemplified in Figure 6 and Table

1 (section 3), CA02 (aspirant member) initiates T01 (membership registration), of which the executor is numbered A01 (register). The result type of T01 is that R01 membership M has been started. The coordination and production bank that belong to T01 are combined in the information bank PB01. In addition, internal actor role A01 gets to know some general information from CPB14, and the personal information about the aspirant member from CPB11.

Listing 1.3. Schema code for Meta Construction Model

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="Transactions">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Transaction" type="TRANSACTION_KIND" maxOccurs="
       unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:complexType name="TRANSACTION_KIND">
 <xsd:sequence>
  <xsd:element name="Tname" type="xsd:string"/>
  <xsd:element name="Initiator" type="ELEMENTARY_ACTOR_ROLE" minOccurs=
      "1" maxOccurs="unbounded"/>
  <xsd:element name="Executor" type="ELEMENTARY_ACTOR_ROLE" minOccurs="
      1" maxOccurs="1"/>
  <xsd:element name="UseInformation" type="INFORMATION_BANK" minOccurs=
      "1" maxOccurs="1"/>
  <xsd:element name="Result" type="DeclaredFactType" minOccurs="1"
      maxOccurs="1"/>
 </xsd:sequence>
 <xsd:attribute name="TransactionID" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="ELEMENTARY_ACTOR_ROLE">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="UseInformation" type="INFORMATION_BANK" minOccurs=
      "0" maxOccurs="unbounded"/>
 </xsd:sequence>
 <xsd:attribute name="ActorID" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="INFORMATION_BANK">
 <xsd:simpleContent>
  <xsd:extension base="xsd:string">
   <xsd:attribute name="BankType" use="optional">
    <xsd:simpleType>
     <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Production"/>
      <xsd:enumeration value="Coordination"/>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:attribute>
   <xsd:attribute name="BankID" type="xsd:string" use="required"/>
  </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="DeclaredFactType">
 <xsd:simpleContent>
  <xsd:extension base="xsd:string">
   <xsd:attribute name="ResultID" type="xsd:string" use="required"/>
  </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```

The MCM is the corresponding metamodel for the instance model described above. Applying the selection rules, we have the information objects that need to be transformed (Table 2). With the guidance of the structure rules, we design the XML Schema (Listing 1.3) as the transformation result of those selected information objects from the MCM.

In order to elaborate the transformation as automatically as possible, an application, which can produce the XML files for DEMO models based on the designed XML schema and input model information, is built. A form (Figure 7) is designed to collect the required information objects from instance CM (Figure 6, Table 1). All the filled information will be processed in the application and structured in a new XML file (Listing 1.4), of which the structure is defined in the above schema (Listing 1.3).

Listing 1.4. The XML document of CM

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Transactions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="C:\Tomcat\webapps\ROOT\demo\CM.xsd">
<Transaction TransactionID="T01">
 <Tname>membership registration</Tname>
 <Initiator ActorID="CA02">
  <name>aspirant member</name>
   <UseInformation BankID="PB01">PB01</UseInformation>
  </Initiator>
  <Executor ActorID="A01">
   <name>registrar</name>
   <UseInformation BankID="PB01">PB01</UseInformation>
   <UseInformation BankID="CPB14" BankType="Production">general data</
       UseInformation>
   <UseInformation BankID="CPB11" BankType="Production">personal data</
       UseInformation>
  </Executor>
  <UseInformation BankID="PB01">PB01</UseInformation>
  <Result ResultID="R01">membership M has been started</Result>
</Transaction>
```

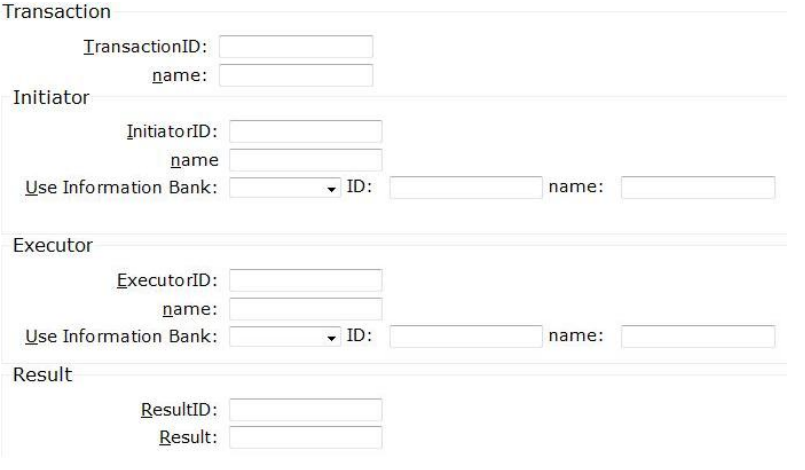


Fig. 7. The information items required for CM

This is one simple example to demonstrate the transformation procedure and show the result of our proposed DEMO transformation. In next section, an evaluation will be made to this transformation result.

6 Evaluation

In this section, we will evaluate the proposed transformation from two perspectives, which are the verification and the usefulness of the transformation result respectively.

6.1 Verification

Verifying the transformation results is of significant importance to the feasibility of the proposed transformation approach. We will present a comparison of the produced exemplary XML documents and their corresponding original DEMO models. The comparison aims to map the information contained in the XML documents with the information from the original models, to see whether the information is completely and precisely maintained during the transformation.

The comparison between the XML document and the original diagram of CM is carried out in the unit of business transaction, including the actor roles that participate in the transaction and the information banks belonging to or used by particular transaction (Figure 6). In section 5 we provide the semantic description and the XML document of this transaction. Here we map them precisely in Table 4.

Clearly seen from the comparison, all the items in the original CM have been correspondingly stored in the XML document, which implies that the

Table 4. Mapping of T01 membership registration

Description of original CM	XML document
T01 membership registration	<Transaction TransactionID="T01"> <Tname>membership registration</Tname>
CA02 (aspirant member) initiates T01 information bank PB01 belong to transaction T01	<Initiator ActorID="CA02"> <name>aspirant member</name> <UseInformation BankID="PB01">PB01</UseInformation> </Initiator>
the executor is A01 (register)	<Executor ActorID="A01"> <name>registrar</name>
information bank PB01 belong to transaction T01	<UseInformation BankID="PB01">PB01</UseInformation>
A01 gets general information from CPB14	<UseInformation BankID="CPB14" BankType="Production">general data</UseInformation>
A01 gets personal information from CPB11	<UseInformation BankID="CPB11" BankType="Production">personal data</UseInformation> </Executor>
information bank PB01 belong to transaction T01	<UseInformation BankID="PB01">PB01</UseInformation>
result type of T01 is R01	<Result ResultID="R01">membership M has been started</Result> </Transaction>

model transformation is completely and correctly maintained during our proposed transformation procedure.

6.2 Usefulness

The XML documents of the DEMO aspect models resulting from the transformation process as introduced in the previous sections, should be of use for several types of applications. As already mentioned in the introduction, the resulting XML documents have added value for simulation tools as well as for tools used to identify business components allowing for information systems to be modeled at a high-level of abstraction. It goes beyond the scope of this paper to show how the transformation results are used for such kind of applications. We refer to [10] for further details on the usage of the transformation results for the identification of business components.

7 Conclusion

This paper discusses an approach to transform DEMO metamodels into XML Schema, allowing to export the semantic data of DEMO models into XML documents. This data can then be extracted by several different applications for further processing.

The approach involves a research about background information on model transformation, a grasp of DEMO metamodel, the definition of transformation rules, the design of XML schema for the DEMO metamodel, the instantiation of the designed XML schema, and the transformation results evaluation. A comprehensive interpretation of the DEMO metamodel was made in XML Schema. We made three transformation rules for the design of XML schema, which specify the content of the proposed DEMO transformation, determine the structure of the expected XML documents, and guarantee the completeness of the selected information and the preciseness of the transformed information. The designed XML schema can be instantiated into a set of XML documents with concrete model information as e.g., from the exemplary case. The produced XML documents are the transformation results. By semantically mapping the information items in the XML documents with the ones in the original aspect models, we verify the transformation results and guarantee the information completeness during the transformation procedure.

Based on this paper, some future works are envisioned. Firstly, reproducing the DEMO diagrams from the transformation results is expected, which provides a more direct verification of the DEMO transformation. Secondly, an automation for combining e.g., graphical user interfaces, used for producing DEMO models, and the tool, used to identify business components, is desired to be investigated in future research. The integration of these tools, based on the XML transformation of DEMO models as presented in this paper, will support designers in generating high-level constructional information system models, by means of business components. Thirdly, the simulation of DEMO models is another direction into which we want to continue our research. Any conceptual

model developed for simulation, will impact all aspects of the subsequent simulation study such as the simulation model development speed, the validity of the model, the experimentation and the confidence based in the model and future reusability of the models. Having an input conceptual model based on theory, as the Ψ-theory DEMO is based on, and having the models transformed into an exchangeable format (XML), lays down a profound opportunity for generating simulation models that can be automatically analyzed. Furthermore the XML format is simulation environment independent, which allows the simulation models to be generated using any simulation environments or tools. Although the simulation is not discussed in details in this paper, it opens up a potential future research, which will be investigated by the authors.

Acknowledgments

This project is supported by the Swiss National Science Foundation (SNSF).

References

1. Hamel, G., Prahalad, C.K.: The Core Competence of the Corporation. In: Strategische Unternehmungsplanung Strategische Unternehmungsführung, Springer, Heidelberg (2006)
2. Dietz, J.L.: Enterprise Ontology Theory and Methodology. Springer, Heidelberg (2006)
3. Dietz, J.L.: The deep structure of business processes. Communications of the ACM 49(5) (May 2006)
4. Albani, A., Dietz, J.L.: Enterprise ontology based development of information systems. International Journal of Internet and Enterprise Management, Special Issue on Enterprise Design and Engineering 7(1) (2011)
5. Albani, A., Dietz, J.L.: The benefit of enterprise ontology in identifying business components. In: The Past and Future of Information Systems: 1976–2006 and Beyond, IFIP 19th World Computer Congress, TC-8, Information System Stream, Santiago de Chile, Chile. IFIP International Federation for Information Processing, vol. 214, pp. 243–254. Springer, Boston (2006)
6. Goknil, A., Topaloglu, Y.: Ontological perspective in metamodeling for model transformations. In: MIS 2005: Proceedings of the 2005 Symposia on Metainformatics, ACM, New York (2005)
7. Koch, N.: Transformation techniques in the model-driven development process of uwe. In: ICWE 2006: Workshop Proceedings of the Sixth International Conference on Web Engineering, ACM, New York (2006)
8. OMG, O.M.G.: Mda guide version 1.0.1 (June 2003)
9. Kurtev, I., van den Berg, K., Aksit, M.: Uml to xml-schema transformation: a case study in managing alternative model transformations in mda. In: Forum on specification and Design Languages (FDL 2003), ECSI, Frankfurt, Germany (2003)
10. Wang, Y.: Transformation of demo models into exchangeable format. Master's thesis, Delft University of Technology (April 2009)

Enterprise Architecture for Small and Medium Enterprise Growth

Dina Jacobs[1,3], Paula Kotzé[1,2,4], Alta van der Merwe[1,2], and Aurona Gerber[1,2]

[1] School of IT, North West University, Vanderbijlpark, South Africa
[2] CSIR Meraka Institute, P.O. Box 395, Pretoria, 0001, South Africa
[3] TriVector, P.O. Box 68753, Highveld 2, 0169, South Africa
[4] School of ICT, Nelson Mandela Metropolitan University, Port Elizabeth, South Africa
dina.jacobs@trivector.co.za,
{paula.kotze,aurona.gerber,alta}@meraka.org.za

Abstract. A key constraint for growing small and medium enterprises (SMEs) is the business skills required to grow the enterprises through the stages of transformation. Criticism against growth stage models for SMEs is of concern, since these models contain the typical knowledge that appeals to managers of small enterprises as guidance in how to manage growth. In this article we propose the SMEAG model to explore the relevance of enterprise architecture (EA) for enhancing existing growth stage models in order to counteract some of this criticism. EA is well-known as a field that claims to manage change and complexity. The rationale to combine the concepts of growth stage models and EA is based on the level of change and complexity associated with the growth of small enterprises into medium enterprises. SMEAG combines the existing growth stage model of Scott and Bruce, the Enterprise Architecture Framework by Hoogervorst, and the EA as Foundation for Business Execution Model by Ross, Weill and Robertson.

Keywords: Enterprise architecture, small and medium enterprises, growth stage models.

1 Introduction

Growing small enterprises to become medium enterprises, with the objective of job creation in South Africa, is a top priority [5]. However, a key constraint is the business skills required to grow the small enterprises through the various stages of transformation. This lack of business skills as constraint is confirmed from a global perspective by Jones [7: p. 1] in his statement "it is recommended that training be provided for all SME entrepreneurs to prepare them for the road ahead and the challenges and crisis that they will inevitable meet along the way". Hanks, Watson et al. [4] also refer to the lack of business skills, although phrased slightly differently: "piloting an organization through the growth process represents a formidable managerial challenge".

The initial assumption may be that there are consolidated growth stage models available for small and medium enterprises (SMEs) to address this lack of business

A. Albani, J.L.G. Dietz, and J. Verelst (Eds.): EEWC 2011, LNBIP 79, pp. 61–75, 2011.

skills. However in a review of relevant material [2, 4, 7, 9-10, 12] there is evidence that this assumption may be questionable, specifically due to the status of such growth stage models for SMEs. In their review of research on small firm growth Davidsson, Achtenhagen and Naldi [2] define growth stage models as a description of the distinct stages of SME growth and the set of typical problems and organizational responses associated with each stage. They noted that authors of review articles on growth stage models for SMEs agree that it is not easy to extract a coherent picture from research, but the inherent complexity of the phenomenon is at least acknowledged. One of the critiques is that the growth stage models tend to assume all SMEs pass inexorably through each stage. A second critique is that growth stage models of SMEs are not sufficiently supported by empirical observation.

This criticisms of growth stage models is of concern since these models typically contain the knowledge that appeals to managers of small enterprises [2, 9].

Enterprise architecture (EA) is widely claimed to be an approach to manage change and complexity [6, 18]. EA not only constitutes a baseline for managing change, but also provides the mechanism by which the reality of the enterprise and its systems can be aligned with management intentions [16]. We argue that EA can contribute towards a solution to the criticism against growth stage models that a small enterprise may not pass through all stages of transformation.

This paper explores using EA to enhance existing SME growth stage models with the objective to provide guidance for SME managers during the transformation process from being a small enterprise to becoming a medium enterprise. The rationale to combine the concepts of growth stage models and EA is based on the level of change and complexity associated with the growth of small enterprises into medium enterprises.

The output of this research is the proposed 'SME EA growth' (SMEAG) model. Experience in industry assisted with developing the SMEAG model through combining theories from the SME growth stage models and EA domains. The SMEAG model is derived by combining the existing growth stage model of Scott and Bruce [14], the Enterprise Architecture Framework by Hoogervorst [6] and the EA as Foundation for Business Execution Model discussed by Ross, Weill and Robertson [13]. The SMEAG model allows for judicious selection of appropriate states and transition during the SME growth process. The proposed SMEAG model is illustrated, using as case study, the operating model of an SME that is in the transformation phase from being a small to becoming a medium enterprise.

The value contribution of the SMEAG model can be summarised as the enrichment of the existing SME growth stage model concept with the following three concepts from the EA domain:

- Replacing the stage concept with a current to future state transition approach.
- The Hoogervorst Enterprise Architecture Framework [6] to indicate the areas of concern, design domains and the architecture principles and standards.
- The Foundation for Business Execution Model [13] to identify the operating model and the level of standardisation and integration required.

The SMEAG model is developed keeping the constraint of resource poverty in the SME world in mind.

Section 2 describes the background by examining the domains of SMEs, growth stage models and enterprise architecture. The proposed SMEAG model is presented in section 3. Section 4 illustrates the application of the SMEAG model in a case study and discusses the value of SMEAG model in the context of this case study. Section 5 concludes with a reference to future research.

2 Background

This section provides the background and the motivation for the research by introducing the relevant domains, namely SMEs, growth stage models and enterprise architecture.

2.1 Small and Medium Enterprises (SMEs)

In order to better comprehend the problem domain, it is necessary to understand the nature of SMEs compared to their larger counterparts. Several factors can play a role, namely:

- The role of SMEs as part of the global and the local (in our case South African) economies. According to Cassell, Nadin et al. [1] approximately 99% of all firms in the EU are SMEs, which employ about 65 million people in total. Globally SMEs account for 99% of business and 40% to 50% of gross domestic product (GDP).
- The reality of resource poverty in the SME world. Welsh and White [15] argued that the very size of a small business creates a special condition, referred to as resource poverty, distinguishing them from their larger counterparts and which requires different management approaches than that followed by larger business.
- Only a small percentage of SME owners envision growing from a small to a medium enterprise. Several studies have shown that across countries, SME growth is not the norm [2]. Most firms start small, live small and die small, and most business founders have modest growth aspirations for their firms. According to Jones [7] the average life cycle of SME's is in the region of five years or less.

The Global Entrepreneurship Monitor (GEM) report [5] identified the need in South Africa to assist small enterprises to grow into medium enterprises and in doing so to stimulate job creation. The GEM research program was initiated in 1997 as a joint venture between academics at London Business School and Babson College in the United States. GEM has grown to a consortium of 64 national teams and is regarded as one of the most important longitudinal studies of entrepreneurship in the world.

From the GEM perspective [5], a number of statistics and statements relevant to SMEs in South Africa can be listed to provide a better understanding of the cause of the problem addressed in this paper. Of the 2.4 million registered companies in South Africa in 2009, 2.2 million were SMEs. SMEs thus play an important part in the economy. Only a small fraction of firms (3.9%) in the start-up phase employ any staff, and only a tiny fraction (<3%) of necessity-oriented businesses create six or

more jobs. The GEM Report also mentioned that formal business require training in skills, such as how to keep records, budget, manage cash flow, maximize trade credit and write a business plan.

2.2 SME Growth Stage Models

Growth stage models are important for SMEs in order to understand. manage and predict problems that might arise during growing the business. The question is whether growth stage models can successfully assist the SME manager that wants to transform the small enterprise in to a medium enterprise.

Both Davidsson et al. [2] and McMahon [9] did comprehensive reviews of literature related to SME growth stage models. According to Davidsson et al. [2], studies of small firm growth are no longer in short supply, but it does not necessarily imply that everything is known about small firm growth. All of the authors of the articles they reviewed commented on the lack of a coherent picture portrayed by reference material. However, there is no evidence in the literature that this identified lack of material is currently addressed by researchers. This is unfortunate because growth stage models represent the type of knowledge small firm managers typically require.

Both Davidsson et al. [2] and McMahon [9] refer to the seminal book by Penrose [11] explaining the two different connotations of growth, namely the amount of growth versus the process of growth. SME growth stage models are related to the process of growth. SME growth is viewed as a series of phases or stages of development through which the business may pass during an enterprise life-cycle.

Massey, Lewis et al. [8] confirmed that the life cycle phenomenon has been found meaningful by SME owner managers. A comprehensive comparison of ten life-cycle models, with particular focus on the life cycle stages and the organizational dimensions used to describe them, is reported in Hanks et al. [4].

All the reviews [2, 4, 8-9] mentioned the justified criticism regarding over-determinism, questionable empirical support and that the stage models tend to assume all SMEs pass through each phase of a growth stage model.

Various models have been proposed specifically addressing the criticism regarding the sequential stages. As an example, Perenyi, Selvarajah et al. [12] proposes a conceptual model with the focus on the transitions between the life cycle stages. These transitions can indicate the development of SMEs, without constraining the model by imposing the sequential nature of the stages. In Hanks et al. [4] it is proposed that each life-cycle stage consists of a unique configuration of variables related to organization context and structure.

The SME growth stage models that focus on generic problems organizations may encounter during growth is, however, valuable for the definition of SME operating models and assisting SME managers to make important decisions [7]. The model by Greiner [3] makes entrepreneurs aware of possible crises and solutions as part of the transformation through the different stages. The model by Scott and Bruce [14], the five stages of which is illustrated in Fig. 1, is based on the model by Greiner.

	Stage 1 Inception	Stage 2 Survival	Stage 3 Growth	Stage 4 Expansion	Stage 5 Maturity
Stage of Industry	Emerging, fragmented	Emerging, fragmented	Growth, some larger competitors, new entries	Growth, shakeout	Growth/shake out or mature/declining
Key Issues	Obtaining customers, economic production	Revenues and expenses	Managed growth, ensuring resources	Financial growth, maintaining control	Expense control, productivity, niche marketing if industry declining
Top Management role	Direct supervision	Supervised supervision	Delegation, co-ordination	Decentralization	Decentralization
Management Style	Entrepreneurial, individualistic	Entrepreneurial, administrative	Entrepreneurial, co-ordinate	Professional, administrative	Watchdog
Organization Structure	Unstructured	Simple	Functional, centralized	Functional, decentralized	Decentralized functional/product
Product and Market Research	None	Little	Some new product development	New product, innovation, market research	Production innovation
Systems and Controls	Simple bookkeeping, eyeball control	Simple bookkeeping, personal control	Accounting systems, simple control reports	Budgeting systems, monthly sales and production reports, delegated control	Formal control, systems management by objectives
Major Source of Finance	Owners, friends and relatives, suppliers leasing	Owners, suppliers, banks	Banks, new partners, retained earnings	Retained earnings, new partners, secured long-term debt	Retained earnings, long-term debt
Cash Generation	Negative	Negative / breakeven	Positive but reinvested	Positive with small dividend	Cash generator, higher dividend
Major Investments	Plant and equipment	Working capital	Working capital, extended plant	New operating units	Maintenance of plant and market position
Product-market	Single line and limited channels and market	Single line and market but increasing scale and channels	Broadened but limited line, single market, multiple channels	Extended range, increased markets and channels	Contained lines. Multiple markets and channels

Fig. 1. Scott and Bruce SME Growth Model [14]

In the Scott and Bruce model the different criteria, such as stage of the industry, key issues, etc., are presented related to each stage, from the inception stage through to the maturity stage. For example, in the first stage, *inception*, the key issues are that of obtaining customers and economic production, which change in the *maturity* stage to that of expense control, productivity, and niche marketing if the industry is declining.

2.3 Enterprise Architecture

Section 2.1 confirmed the importance of SMEs to contribute to job creation considering the large number of SMEs that is part of the economy. Three of the challenges derived from the discussion in section 2.1 are the phenomenon of resource poverty, the small number of SMEs that are interested in growth to become a medium enterprise, and lack of skills of SME managers to transform a small enterprise into a medium enterprise.

The question is whether growth stage models can successfully assist the SME manager that wants to transform the small enterprise to a medium enterprise. Section 2.2 mentioned that growth stage models are criticized. One of the key reasons for this criticism is that all the enterprises do not pass through all the stages in a specific sequence following all the stage criteria for a specific stage. It was also noted that the transformation from a small to medium enterprise is complex with high change impact.

From the perspective of change and complexity EA is considered as a discipline that could contribute to the solution to assist the SME management during the growth of a small enterprise. Investigating the relevance of EA to compliment growth stage models is therefore relevant.

The four EA concepts that are specifically considered as part of the development of the SMEAG model are briefly presented in the remainder of this section, namely:

- The relevance of EA considering the change and complexity associated with SME growth.
- The introduction of the state transition approach versus a stage based approach.
- The Hoogervorst EA Framework [6].
- The Foundation for Business Execution Model [13].

The Hoogervorst EA Framework [6] is defined as the expression of aspects (areas of concern and design domains) that are considered relevant and must be addressed by the architecture to be defined. EA is defined as a coherent and consistent set of principles and standards that guides enterprise design [6].

Hoogervorst [6] states that enterprise engineering enables enterprise change and adaptation, with EA providing the guidance for the design in order for the enterprise to operate as a unified and integrated whole. Zachman [17] also discussed EA as an approach to manage change and complexity by emphasising the state transition concept of the change from a current state (as-is) to a future state (to-be) perspective.

Ross et al. [13] proposed the Foundation for Business Execution Model where enterprises have to define their operating model and define the processes and infrastructure critical to their operations (i.e. their EAs). The model describes how an enterprise can thrive and grow. In order to grow you need to understand the relevance of process standardization and process integration as part of the transformation process. Process

standardization delivers efficiency and predictability across the company and integration links the efforts of organizational units through shared data.

The operating model determines the level of standardization and integration required. Four different types of operating models are described [13], namely diversification (low standardization and low integration), coordination (low standardization and high integration), unification (high standardization and high integration) and replication (high standardization and low integration). For each of the operating models a core diagram is proposed [13]. Fig. 2, for example, presents the proposed *replication* core diagram. When designing a *replication* core diagram, one starts with the identification of the key processes to be standardized and replicated across the business units. The next step is to identify the technologies automating those key processes. It is not necessary to include the data and customers as part of the core diagram as integration is not a requirement to support growth as part of the replication operating model.

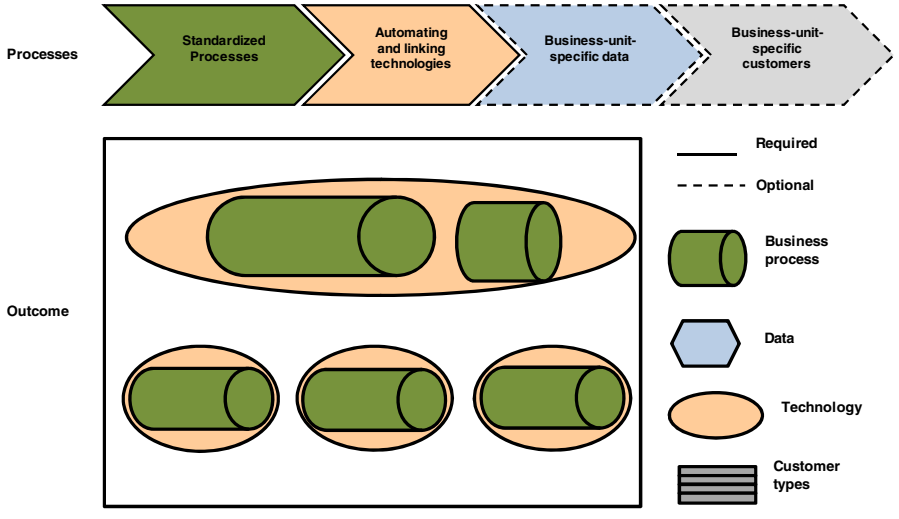


Fig. 2. Replication core diagram (adapted from Ross et al. [13])

3 The Proposed SMEAG Model

In this section we propose the SMEAG model, where the objective is to provide a model for SME managers that are involved with the transformation of a small enterprise into a medium enterprise to assist them during the growth process. The abbreviation SMEAG evolved from the different concepts included in the proposed model, namely *Small Medium Enterprise + Enterprise Architecture + Growth*.

The SMEAG model is an enhancement of the Scott and Bruce growth stage model [14] incorporating:

- the EA principle of a current and a future state,
- the Hoogervorst EA framework [6] concept describing the areas of concern, the domain and the enterprise architecture principles and standards, and

- the operating model and core diagram concepts from the Foundation for Execution model of Ross et al. [13].

The growth stage model by Scott and Bruce [14] was selected since it highlights the typical decision making points in the transformation from a small to medium enterprise.

The Scott and Bruce growth stage model (Fig. 1) was adapted not to pass through the stages sequentially and phase by phase, but rather based on the identification of the current state to the future state concept advocated by EA.

The Hoogervorst EA framework [6] was selected to incorporate EA concepts including the areas of concern, EA design domains and the EA principles and standards.

The Foundation for Business Execution Model [13] was selected as EA model for inclusion in the SMEAG model to support the applicable operating model. It is the potential value of the operating model to address growth that makes this model applicable to the SME manager. Not only will it position the importance of the selection of an appropriate operating model, but also the level of process standardization and process integration required to support the growth based on the operating model.

The SMEAG model, as illustrated in Fig. 3, consists of three components:

- The SMEAG Fact Sheet.
- The SMEAG Work Sheet.
- The Operating Model Core Diagram.

These components are described in more detail in the following three sub-sections.

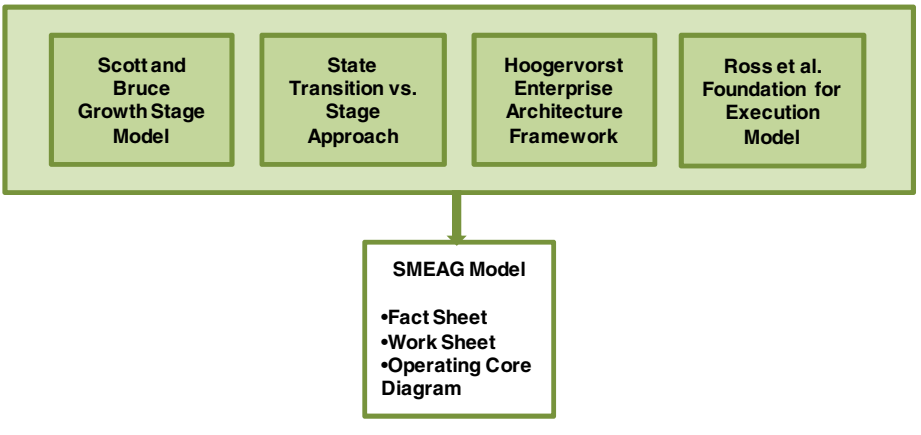


Fig. 3. SMEAG Model

3.1 The SMEAG Fact Sheet

The SMEAG Fact Sheet is a generic accelerator using EA to enhance the SME growth stage models that is pre-populated and available as accelerator for the SME management.

There are two aspects that relates to the SMEAG Fact Sheet. The first is the structure, referring to the relationships between the *Area of Concern*, the typical options to

consider describing the *State*, the *Design Domain* and the *Architecture Principles and Standards*. This generic structure is derived from the Scott and Bruce Growth Stage Model [14] (Fig. 1) and the Hoogervorst Framework [6].

The second aspect is the content of the SMEAG Fact Sheet, as illustrated with examples in Fig. 4. The first row, *Organization Structure*, has its origin in the Scott and Bruce growth stage model [14] , and the second row, *Operating Model* is a contribution from the Foundation for Business Execution Model [13]. The *Area of Concern* is identified from either a growth stage model or an EA model. The various *States* are identified from the various sources and then the *Design Domai*n is allocated with guidance from Hoogervorst [6]. The *Architecture Principles and Standards* are either sourced from EA models or developed by EA experts. The preparation of the SMEAG Fact Sheet is not merely a 'concatenation' of the various sources, but rather an analysis, contextualisation and alignment of the information from the various sources.

Area of Concern	State Options	Design Domain	Architecture Principles and Standards
Organization Structure	Unstructured Simple Functional centralized Functional decentralized Product/... decentralized	Organization Technology	* Grouping of activities (units) must create minimized cross-boundary relationships. *Process design must address delegation of coordination activities explicitly. *Collaboration services must be made available.
Operating Model	Diversification Co-ordination Replication Unification	Business Information Technology	*Key processes to be standardised. *Multiple customer interaction channels must operate transparently (inter functionally) *Informational data may have only one authorizing source. *Customer data must be available from one unified source. *Redundant data entry about the same data is not allowed.

Fig. 4. SMEAG Fact Sheet

3.2 The SMEAG Work Sheet

The second component of the SMEAG model is the SMEAG Work Sheet. In contrast with the SMEAG Fact Sheet the SMEAG Work Sheet is dependent on the input of the SMEAG management to complete the following four steps:

1. The review and extension of the SMEAG Fact Sheet with SME specific 'areas of concern', if available.
2. The selection of the 'areas of concern' from the SMEAG Fact Sheet applicable to the SME.
3. The identification of the current state per area of concern from the SMEAG Fact Sheet, as well as the future state if relevant.

4. To determine the actions required for the transition from the current state to the future state.

The SMEAG Work Sheet is an input for the business plan prepared by the SME management for each financial year. An example of a Work Sheet is illustrated in Fig. 7 when we discuss the case study.

3.3 The SMEAG Operating Core Diagram

The third component of the SMEAG model is the Operating Model Core Diagram to guide the SME manager through the standardization and integration of processes and identification of the enabling technology. This step involves the incorporation of the Foundation for Business Execution Model [13]. At this stage it is necessary to confirm the operating model for growth (diversification vs. coordination vs. unification vs. replication.)

4 Case Study

This section illustrates the use of SMEAG using a case study. Section 4.1 provides the case study background. Section 4.2 highlights the stage vs. state problem and section 4.3 applies SMEAG.

4.1 Case Study Background

The case study is based on an SME that is growing from a small enterprise into a medium enterprise. The nature of the underlying business conducted by the small enterprise is that of a 'consulting practice' with a narrowly defined service range. The number of full time employees is around 35 and the number of sub contractors varies between 10 and 20.

The SME's management wants to understand the areas of concern and wish to identify the initiatives to be included in the business plan to manage the growth from a small to medium enterprise deliberately.

During 2010 the SME developed an operating model with one of the objectives the growth of the enterprise from a small into a medium enterprise. The growth model for 2011 is based on the replication of new pipelines and, although not clearly stated as part of the 2010 operating model, the *replication model* [13] was found a good fit to describe the growth model. A brief overview of the 2010 operating model of the SME is included in Fig. 5.

4.2 Stage vs. State Problem Confirmation

The SME's initial problem was that it was not possible to determine the current and future 'stage' of the SME using growth stage models as guideline.

Using the 2010 operating model (Fig. 5), the current and future states of the SME were mapped according to the Scott and Bruce model (the model is illustrated in Fig. 1). The outcome of the current and future states mapping for the case study SME is illustrated in Fig. 6. The current state varies between Stage 2 and Stage 4, and the

Focus Area	Current State	Future State
Ownership: Legal structure	One registered entity	No change
Ownership: Shareholding	Current shareholding weighted towards non managerial interest.	Restructuring of shares into individual management as opposed to investment hands.
Cash generating capability	Solid	No change
Operating Model	Although different "markets" exist within the definition of the current business focus the business runs as one consolidated whole	The end-state envisages the dismantling and restructuring of the entire organization so as to: Stabilize end state revenue. Refocus the business mix so as to achieve higher margins. Maintaining overheads. Changing financial reporting.
Operating Model: Dividend Policy	No defined dividend policy.	Dividend policy in conjunction with risk and performance management system.
Operating Model: Business Definition	Company plays in the bottom segment and margins are at the lower level of the possible range.	The end-state envisages a drift to higher margin business within the current business definition and a diversification into "efficiency" originated process work.
Operating Model: Performance Management	The organization design (and the financial support system) is not geared to report on divisional /project profitability.	A performance management system based on consistency of rewards versus roles and a clearly articulated and motivational reward system.
Risk Management	Retention of cash and a mixed of permanent and variable resources.	Limitation of current model turnover. A mix of fixed versus variable resources. Retention of cash.
Risk: Contingent liabilities or area of future financial risk	None	No change
Organizational Structure: Corporate Governance	Multitude of roles played by certain individuals.	A more formalized governance structure.

Fig. 5. 2010 Operating model of SME to be used as case study

future state between Stage 3 and Stage 5 of the Scott and Bruce model. For four of the criteria there is no difference between the current and future states.

This mapping illustrates why the SME had a problem to determine its current and future 'stage' according to the guidelines of growth stage models. The mapping illustrates that an enterprise is not necessarily in the same stage for all criteria when growing, i.e. an SME does not necessarily progress sequentially and simultaneously through all the criteria of a stage. The enterprise thus may not gain any value by moving automatically to the next stage for of all the criteria, as proposed by growth stage models.

The proposed SMEAG model, suggesting a possible way to address this problem, was next applied to the case study.

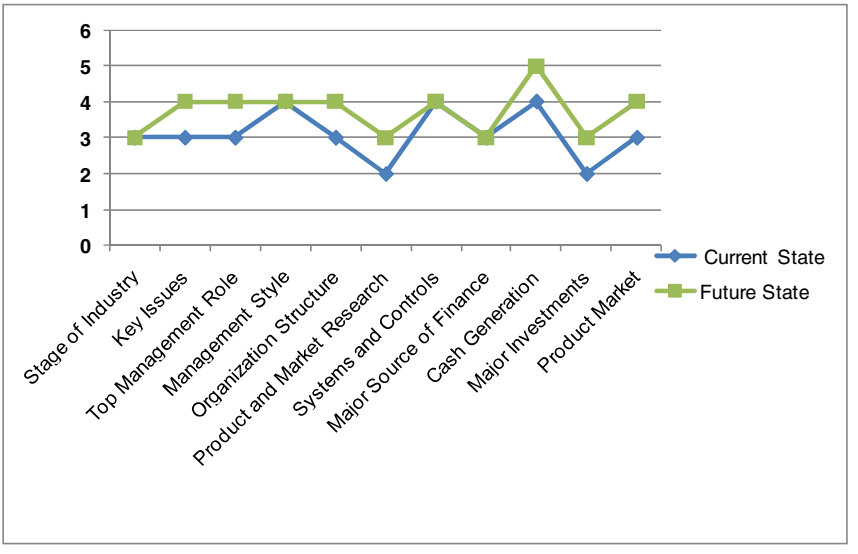


Fig. 6. Current and Future State (Case Study)

4.3 SMEAG Model

The case study SMEAG model is based on the 4 steps described in section 3.2. Step 1 is to review the SMEAG Fact Sheet with the 2010 Operating Model of the SME (Fig. 5). An example is included as illustration in Fig. 7.

Area of Concern	State Options	Design Domain	Architecture Principles and Standards
Dividend Policy	No dividend policy Dividend policy	Business	* Minimum 10% of turnover available for working capital before dividend is considered.

Fig. 7. Example SMEAG Extended Fact Sheet (Case Study)

The second step is to complete the SMEAG Work Sheet as described in section 3.3. The third step, adding the required actions are also included in Fig. 8 to illustrate the outcome.

Area of Concern	2010 State (Current)	2011 State (Future)	Actions
Organization structure	Functional centralized	Functional centralized	Review process design to ensure delegation of coordination activities is explicitly addressed.
Operating model	None	Replication	Key processes to be standardised.

Fig. 8. Example SMEAG Work Sheet (Case Study)

The final step is to prepare the replication core diagram. The replication core diagram is representing the key processes to be standardized as well as the enabling technology. The case study described in this paper is based on replication as operating model and Fig. 9 illustrates the replication core diagram for this case study. Since it is a replication model only processes and technology are included in the diagram (integration is not a not a requirement to support growth as part the replication operating model). The identification of the key processes was done during a work session using reference models as accelerator.

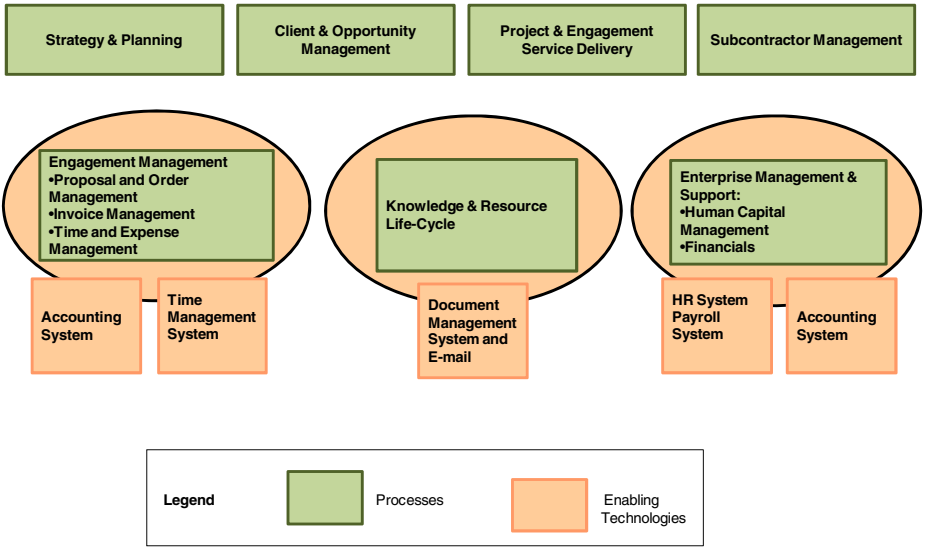


Fig. 9. Case study replication core diagram

4.4 Discussion

From the perspective of the SME management the SMEAG model successfully addressed the 'stage' problem in the case study by replacing it with the state based approach. The SMEAG Fact Sheet was successfully used to populate the SMEAG Work Sheet. The final outcome is the 2011 Business Plan and the SMEAG Work Sheet contributed to a more complete description of the areas of concern. The value of the replication core diagram is seen as the first step to standardize the key processes for roll-out to the different pipelines. Based on the architectural principles and standards the SME management are asking for the integration of the systems to have a single source for information. A key benefit is that accelerators are making it a feasible model from a SME resource poverty perspective, considering that the number of hours required from SME management is relatively low and the cost implication a minimum.

5 Conclusion

The primary objective of the SMEAG model is to provide guidance for SME managers during the transformation process of growing from a small into a medium

enterprise. In the case study that was used to illustrate the use of the SMEAG model, the state based approached was successfully used to address the stage based problem. There is thus evidence that enterprise architecture is relevant to complement existing SME growth stage models. Not only does EA address the gap regarding the configuration of the various current and future states per area of concern, it also enhances the model to have a better understanding of the different operating models for growth and the importance of the standardization and integration of processes.

The key contributions of the SMEAG model are:

- The positioning of the current state versus future state concept.
- The positioning of EA regarding its principles and standards against those of growth stage models..
- The integration of the operating model core diagram to identify process standardization and integration.
- The packaging of the model to make it a practical tool for SME managers.

The outcome of this research is the first version of the proposed SMEAG model. The next iteration of development will focus on the completeness of the model, the interdependencies between the areas of concern, as well as the options indicating the state per area of concern in the SMEAG Fact Sheet. The content of the SMEAG Fact Sheet will also be verified and extended against more growth stage models and other EA models.

References

1. Cassell, C., Nadin, S., Gray, M., Clegg, C.: Exploring human resource management practices in small and medium sized enterprises. Personnel Review 31, 671–692 (2002)
2. Davidsson, P., Achtenhagen, L., Naldi, L.: Research on Small Firm Growth: A Review (2005)
3. Greiner, L.E.: Evolutions and revolutions as organizations grow. Harvard Business Review 50, 37–46 (1972)
4. Hanks, S.H., Watson, C.J., Jansen, E., Chandler, G.N.: Tightening the life-cycle construct: a taxonomic study of growth stage configurations in high-technology organizations. Entrepreneurship Theory and Practice 18, 5–29 (1993)
5. Herrington, M., Kew, J., Kew, P.: Tracking Entrepreneurship in South Africa: A GEM Perspective (2010)
6. Hoogervorst, J.A.P.: Enterprise Governance and Enterprise Engineering. Springer, Heidelberg (2009)
7. Jones, N.: SMEs Life Cycle – Steps to Failure or Success?
8. Massey, C., L.ewis, K., Warriner, V., Harris, C., Tweed, D., Cheyene, J., Cameron, A.: Exploring firm development in the context of New Zealand SMEs. Small Enterprise Research: The Journal of SEAANZ 14, 1–13 (2006)
9. McMahon, R.: Stage Models of SME Growth Reconsidered (1998)
10. Miller, D.: The Genesis of Configuration. Academy of Management Review 12, 686–701 (1987)
11. Penrose, E.: The Theory Oxford University Press, Oxford (1959)

12. Perenyi, A., Selvarajah, C., Muthaly, S.: The Stage Model of Firm Development: A Conceptualization of SME Growth. In: Proceedings of Regional Frontiers of Entrepreneurship Research 2008: 5th International Australian Graduate School of Entrepreneurship (AGSE) Entrepreneurship Research Exchange, Australian Graduate School of Entrepreneurship, Swinburne University of Technology, Melbourne, Victoria (2008)
13. Ross, J., Weill, P., Robertson, D.: Enterprise Architecture as Strategy Creating a Foundation for Business Execution. Harvard Business School Publishing, Boston (2006)
14. Scott, M., Bruce, R.: Five stages of growth in small business. Long Range Planning 20, 45–52 (1987)
15. Welsh John, A., White, J.F.: Small business ratio analysis: a cautionary note to consultants. Journal of Small Business Management, 20–23 (1981)
16. Whitman, L., Ramachandran, K., Ketkat, V.: A taxonomy of a living model of the enterprise, pp. 848–855. IEEE Computer Society, Los Alamitos (2001)
17. Zachman, J.A.: The Zachman Framework for Enterprise Architecture: Primer for Enterprise Engineering and Manufacturing. Zachman Framework Associates (2006)
18. Zachman, J.A.: The Zachman Framework™: The Official Concise Definition. Zachman International (2008)

A Critical Investigation of TOGAF – Based on the Enterprise Engineering Theory and Practice

Jan L.G. Dietz and Jan A.P. Hoogervorst

Delft University of Technology,
{j.l.g.dietz,j.a.p.hoogervorst}@tudelft.nl

Abstract. TOGAF (The Open Group Architecture Framework) is growingly considered to be the de facto standard way of working for the development and deployment of modern IT systems in enterprises. A major characteristic of modern IT systems, as opposed to the ones in the past, is that they are an integral part of the total enterprise, and effectively support some part of the enterprise's activities. Consequently, they must be developed with unity and integration in mind. Business development, organization development, and IT systems development cannot be addressed anymore as unrelated subjects. In this paper, the authors report on an investigation of TOGAF (version 9) regarding the extent to which it satisfies the indispensable requirement of unity and integration. The conclusion is that TOGAF fails to do this, as it fails to achieve several other ambitions. The main cause of these failures seems to be the lack of a sound and appropriate theory. In carrying out the investigation, the authors have based themselves on the enterprise engineering theory, as well as on extensive practical experiences.

Keywords: TOGAF, Enterprise Architecture, Architecture Framework, Enterprise Engineering.

1 Introduction

The track record regarding successfully implementing strategic initiatives is rather poor. Some publications speak about less than 10% success rate [Mintzberg 1994]. This rather low figure compares with other sources. According to Kaplan and Norton, many studies prove that between 70% and 90% of strategic initiatives fail, meaning that the expected result is not achieved [2004]. Studies concerning a specific strategic domain, such as total quality management, business process reengineering, customer relationship management, or mergers and acquisitions, similarly report high failure rates. All too often, failures are conveniently attributed to unforeseen or uncontrollable external events. However, strategic failure is seldom the unavoidable result of an inadequate strategy, but mostly the avoidable consequence of inadequate operationalization of the strategic intent. A plethora of literature indicates that a (if not the) core reason for strategic failures is the lack of coherence and consistency among the various enterprise aspects, which precludes it to operate in a unified and integrated manner [Kotter 1995, Nadler and Tushman 1997, Pettigrew 1998, Doucet et al. 2009, Leinwand and Mainardi 2010]. We contend that a unified and integrated enterprise

A. Albani, J.L.G. Dietz, and J. Verelst (Eds.): EEWC 2011, LNBIP 79, pp. 76–90, 2011.

does not occur incidentally, but must be intentionally *designed*. A McKinsey publication confirmed this observation: rather than the traditional management focus on ad-hoc structural changes, acquisitions or competition, "they would be better off focusing on organizational design" [Bryan and Joyce 2007].

To ensure enterprise unity and integration many mutually related aspects have to be taken into account. In order to master this enormous task, many authors argue that the system approach is the only way to address the core problem effectively, hence to study and develop enterprises as systems [Bertalanffy 1969, Bunge 1979, Gharajedaghi 1999, Rechtin 2000]. Ackoff therefore argues that the high rate of failing strategic initiatives mentioned previously, is the consequence of the initiatives being fundamentally anti-systemic [Ackoff 1999].

The Open Group Architecture Framework (TOGAF, version 9) is positioned as an approach for developing enterprises and/or enterprise IT systems [The Open Group 2009a]. TOGAF correctly acknowledges that developing entails a design perspective. Therefore, we consider TOGAF to be a candidate methodology in developing enterprises. In this article we will investigate TOGAF in this respect, specifically assessing how the design perspective is operationalized in a systemic approach for ensuring enterprise unity and integration. The authors have based their analysis on the current enterprise engineering theory [Albani & Dietz 2010, Albani & Terlouw 2010, Aveiro et.al. 2010a, Aveiro et.al. 2010b, Barjis et. al. 2009, Dietz 2006, Dietz 2008, Hoogervorst 2009, Jong & Dietz 2010, Nuffel et. al. 2009, Ven & Verelst 2009], as well as on extensive practical experiences[1].

The article is structured as follows. In section 2, the foundations are presented and discussed that are considered appropriate and imperative for a sensible investigation of the claims that TOGAF apparently makes. On these foundations we formulate, in section 3, seven assessment criteria and we assess TOGAF on each of these criteria. In section 4, we summarize the conclusions of our research.

2 The Theoretical Basis for Enterprise Design

As the basis for assessing TOGAF in a profound and thorough way we will present and discuss in this section a theoretical outline regarding the development of systems, including enterprises. The basic paradigm we adopt is that enterprises are purposefully designed systems, in accordance with the Enterprise Engineering Manifesto[2]. We consider this outline suitable for assessing TOGAF since it covers all issues that are dealt with by TOGAF.

2.1 The Notion of System

Various system definitions exist. Jackson sees a system as "a complex whole the functioning of which depends on its parts and the interaction between these parts" [2003, p.3]. Maier and Rechtin define a system as "a set of different elements so connected or related as to perform a unique function not performable by the elements alone" [2002]. Von Bertalanffy speaks of "a set of elements standing in interrelation

[1] The reader is referred to www.ee-institute.com for further information.

[2] http://www.ciaonetwork.org/publications/EEManifesto.pdf

among themselves and with the environment" [1969, p.252]. It appears that there are basically two kinds of system notions: teleological and ontological [Dietz 2006]. The *teleological* system notion is concerned with the function or purpose of the system. This notion is suited and perfectly adequate for using or controlling a system (whereas the ontological notion is not). The associated kind of model is the *black-box model*. Actually, the teleological system notion is equal to a black-box model of the system. This can easily be understood if one recognizes that purpose (or function) is not a system property but a relationship between a system and a stakeholder. For example, every car driver has a black-box model of the behavior of the car he or she drives. Black-box models can be decomposed by means of functional decomposition. Usually, a car driver also has some functional decomposition in his or her mind. It is important to note that black-box models are fundamentally subjective.

The *ontological* system notion regards the construction and operation of a system. It is independent of and therefore indifferent to the function or purpose that one assigns to the system. This notion is suited and perfectly adequate for building and changing systems (whereas the teleological notion is not). The associated kind of model is the *white-box model*. White-box models can be decomposed by means of constructional decomposition. Note that there is only one constructional decomposition of a system.

Based on the rigorous systemic ontology of Mario Bunge [Bunge 1979] we apply the next ontological notion of system [Dietz 2006]. A system is a tuple <C,E,P,S>, where:

C : *composition*: a set of system elements of one and the same system category.
E : *environment*: a set of elements of the same category as the elements of *C*.
P : *production*: the products or services that *C* delivers to *E*.
S : *structure*: the mutually influencing bonds among the elements of *C*, and between the elements of *C* and the elements of *E*.

The concept of system kind or category is crucial for deeply understanding the ontological system notion. Many system categories can be identified, for example biological, chemical, electrical and social. Note that the system definition provided above defines a homogeneous system. The most interesting systems, however, are heterogeneous systems, like cars and enterprises. They are constructions of homogenous systems. The unification and integration of a number of homogenous systems into a heterogeneous system is by no means trivial. That is why unity and integration are core concepts within the system approach [Gharajedaghi 1999].

2.2 The Notion of Architecture and Architecture Framework

So, the systems of our interest are designed systems, like IT systems and enterprises. The design of these systems is by its very nature not 'incidental'. Instead it is a goal-directed process in which the designer needs guidance. One explicit kind of guidance is provided by the system requirements, both the functional requirements and the constructional ones. However, in general these requirements keep an amount of design freedom left that the designer has to cope with somehow. He needs additional, normative, guidelines. Often such guidelines are kept implicit, even to the extent that designers are not aware of them. We fully agree in this respect with Ulrich's critical

system heuristics, arguing that the normative aspects of system design must be made explicit [In: Jackson 2003]. Architecture provides the answer to this question, as it has done since time immemorial in constructional engineering.

Conceptually, architecture thus can be defined as the normative restriction of design freedom [Dietz 2009]. Practically, it consists of *a coherent and consistent set of principles and standards that guide system design* [Dietz 2009, Hoogervorst 2009]. Since architecture guides system design, devising architecture should take place pertinent to relevant system aspects as presented by an architecture framework. We conceive an architecture framework thus as a conceptual structure for architecturing, i.e. for devising architectures. Three dimensions play a role in the process of architecturing [Dietz 2009]:

System kinds, which we will identify by *S*
Design domains, labeled as *D*
Areas of concern, labeled as *A*

These three dimensions seem to be necessary and sufficient. Examples of system kinds are organizations, information systems, and IT systems. The second dimension is the *design domain* to which a design principle belongs. Examples of principles are: 'process control must be separated from process execution', or 'network access must be based on authentication and role-based authorization'. Building on the systemic foundations, as discussed in Section 2.1, we distinguish between two domains on the highest level of detail: function and construction. The third dimension is the *area of concern* that a design principle addresses. Whereas the first two dimensions (system kind and design domain) are timeless, the third one is not. It reflects the, generally time-dependent, focal interests of stakeholders. Examples of areas of concern are security, compliance, privacy, agility, and user-friendliness. They serve to organize the total set of design principles of an architecture into subsets that correspond to the interests of the distinct stakeholders. These subsets need not be disjoint. Put differently, areas of concern may overlap. It means that a particular design principle regards several interests and therefore addresses several areas of concern.

An architecture framework can thus be presented symbolically as a triplet <*S*,*D*,*A*>. It is a conceptual structure related to one or more system kinds, a number of areas of concern, and a necessary and sufficient set of design domains [Hoogervorst 2009]. As said, the highest level distinction between design domains concerns the distinction between the system *function* and the system *construction*. Within this distinction, further detailing is evidently required; it refers to the specialization of design domains associated with more detailed observations. Such specialization thus creates a certain order whereby a more detailed design domain is subordinated under the next higher design domain, in the way that, for example, the design domain 'engine' is subordinated under the overall design domain 'car', and 'piston' in turn is subordinated under the domain 'engine'. This is an important condition for safeguarding coherence and consistency, which has been emphasized previously as an important objective of defining architecture. Establishing the necessary and sufficient (complete) set of design domains can be a daunting task for complex systems, such as enterprises. Knowledge and experience of the architect concerning the system kind in question is obviously crucial.

2.3 The Generic System Development Process

Figure 1 exhibits the so-called Generic System Development Process (GSDP) [Dietz, 2008], which applies to the development of systems of any kind. Taking the function perspective, one `sees' the function and the (external) behaviour of a system; the corresponding type of model is the black-box model. Taking the construction perspective, one `sees' the construction and the operation of a system; the corresponding type of model is the white-box model. In developing a system both the function perspective and the construction perspective are relevant. The GSDP identifies the most basic steps in a development process.

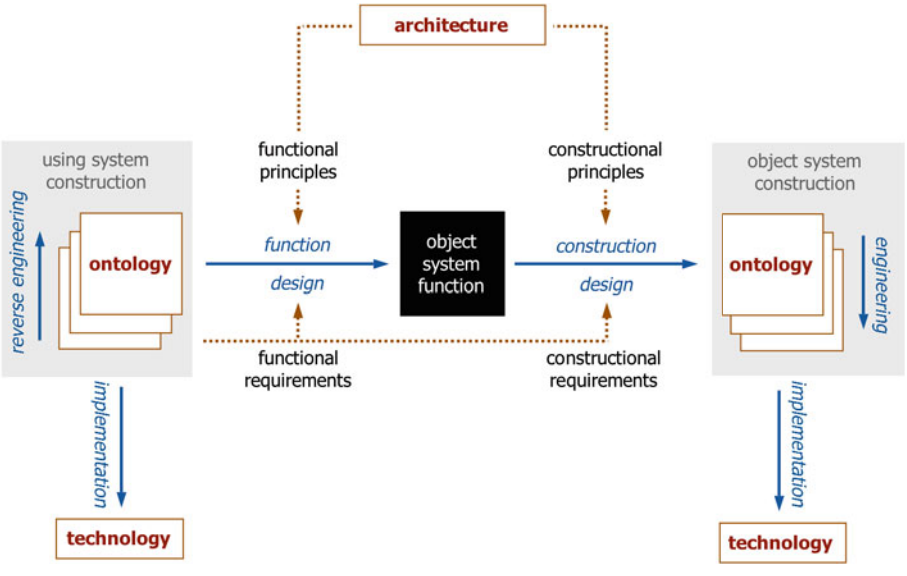


Fig. 1. The Generic System Development Process

The starting point is the need by some system, called the *using system* (US), for a supporting system, called the *object system* (OS). A clear distinction between the US and the OS is often neglected, leading to blurred discussions about the functionality of the OS. In the GSDP, the US is the stable starting point for the development process. One must have an appropriate understanding of the US in order to successfully design an OS. By nature, this understanding must be constructional understanding, since it is the construction of the US that is going to be supported by the function of the OS. So one starts with conceiving a white-box model of the US. Preferably, this model is an ontological model [Dietz, 2006]; otherwise one can easily become confused by irrelevant implementation issues of the US. From the white-box model of the US one determines the functional requirements for the OS (*function design*). These requirements are by nature formulated in terms of the construction and operation of the US. Moreover, and consequently, they need to be fully independent of the construction of the OS. The next basic design step is to devise specifications for the construction and operation of the OS, in terms of a white-box model of the OS (*construction design*).

For this design phase, the US may provide constructional requirements, often also called non-functional requirements. They mostly regard performance and quality aspects. A thorough analysis of the white-box model of the OS must guarantee that building the OS is feasible, given the available technology. The GSDP also includes the important experience from practice that designing is an iterative process: the final result of every design process is (or should be) a balanced compromise between reasonable functional requirements and feasible constructional specifications [Dietz, Albani 2005]. For the sake of simplicity, however, we left it out from the figure.

In addition to the functional and constructional requirements, there may be functional and constructional principles respectively. These design principles are the operational shape of the notion of architecture, as discussed in Section 2.2. They generally hold for a class of systems. An example of a functional principle is that man-machine dialogs must comply with some standard. An example of a constructional principle is that the applications must be component-based. Ideally the construction design phase results first into an ontological model of the OS, i.e. a white-box model that is completely independent of its implementation. Gradually this ontological model is transformed into more detailed (and more implementation dependent) white-box models, the last one being the implementation model. This process is called implementation design or just engineering. If the OS is a software application, then the implementation model would be the source code in some programming language. The act of implementing consists of assigning appropriate technological means to the implementation model, e.g. running the source code on an appropriate platform.

2.4 Enterprise and Enterprise Architecture

Building on the theoretical foundations that have been discussed in Sections 2.1 thru 2.3, we will now provide precise and consistent definitions of the notions of enterprise and of enterprise architecture. Enterprises are goal-directed social entities that are designed as deliberately structured activity systems linked to the external environment [Daft 2001]. All enterprises face the fundamental issue of indispensable *differentiation* (the creation of specific tasks) on the one hand, and establishing *unity and integration* (the realization of coherence and consistency in task execution) on the other hand [Lawrence Lorsch 1967]. Certain cooperative interaction patterns must therefore necessarily exist between human actors for collectively realizing the enterprise purpose and function. Enterprise performance is thus ultimately the result of social interaction between human actors who enter into, and comply with, commitments concerning the execution of tasks. Obviously, an enterprise design theory and methodology must address aforementioned issue effectively. For that we adopt the so-called Ψ(psi)-theory: Performance through Social Interaction [Dietz 2006], which is grounded in the view that the collaborative interaction patterns between human actors in enterprises are constituted by coordination activities, which are based on mutual communication. These notions form the basis for the so-called speech/act theory, or the language/action perspective on the design of cooperative work [Winograd and Flores 1987]. Thus language is seen "as the primary dimension of human cooperative activity" [Winograd 1988]. Within this perspective, the focus is on communicative patterns that constitute the mutual coordination, since people act through language.

Individuals within enterprises fulfill actor roles (manifesting the differentiation in task execution), whereby basically two types of activities are performed: (1) *production*

activities and (2) *coordination activities* [Dietz 2006]. Production activities can yield a material or immaterial result. Material production has to do with manufacturing, storage or the transport of goods for example. Immaterial production concerns decision-making, granting something, sentencing a person by a judge, appointing a person in a function and so on. Coordination activities concern the communicative actions mentioned above pertinent to entering into and complying with commitments about production activities. Coordination activities are therefore always linked to production activities. The enterprise is then seen as a "network of commitments" [Winograd and Flores 1987, p.150]. Coordination and production activities come in universal patterns, called transactions. An enterprise process is thus a structure of causally related transactions.

Based on the general systems development process discussed before, the first step in designing an enterprise is establishing the implementation-independent (ontological) models, which are based on, and reflect, the essential transactions of the enterprise [Dietz 2006]. Discussing these models falls outside our current scope. For now we like to stress that the implementation-independent nature of the models substantially reduces the complexity, hence the effort to comprehend the enterprise. Moreover, the models precisely depict the essential communicative – hence collaborative – patterns between the different actors in the enterprise that collectively depict the whole essential enterprise operation. The issue of differentiation on the one hand, and unity and integration on the other, is thereby formally addressed at the essential, implementation-independent level. Additionally, the explicit modeling of the coordination activities of a transaction allows the precise definition of operational rules guiding these activities.

After having defined the implementation-independent models, further design of the enterprise system (*S*) needs to take place in order to devise construction models that can be implemented. Design takes place in various design domains (*D*), guided by architecture for addressing areas of concern (*A*) and further ensuring unity and integration. As indicated previously, there are basically two main categories of design domains: *functional* and *constructional* ones. Various labels are used in the literature to identify areas where (some form of) design should take place, such as business, information, data, application, infrastructure, or technology. The distinction between function and construction is virtually never made explicitly. With reference to aforementioned distinction, we will identify design domains comprehensively enterprise-wide. Subsequent design pertinent to these design domains (guided architecture) transforms or complements the implementation-independent ontological models in construction models that ultimately can be implemented. We propose four main enterprise design domains [Hoogervorst 2009]:

Business. This design domain concerns the enterprise *function*, having to do with: (1) the elements of the enterprise environment, such as customers, suppliers, business partners, or stakeholders, (2) the products and services the enterprise delivers to its environment, and (3) the relationships between them, like sales and communication channels. In fact, all these topics concern sub (functional) design domains within the main business design domain.

Organization. The organization design domain concerns the internal arrangement of the enterprise for delivering the enterprise's function. It includes sub design domains like processes, employee behavior, enterprise culture, management/leadership

practices, and various structures and systems, such as concerning accounting, purchasing, payment, or employee remuneration and evaluation. These are all topics that must be formally brought within the enterprise design perspective [Luthans 1992, Daft 2001, Schein 2004]

With the label 'organization' the main enterprise *construction* design domain can be identified. In view of information as a crucial production factor, rather than seeing the design domain 'information' as part of the design domain 'organization' we will consider the design domain 'information' as a separate main constructional design domain. Similar considerations hold for the design domain '(information) technology'.

Information. Many informational aspects play a role, such as the structure and quality of information, the management of information (gathering, storage, distribution), and the utilization of information. These topics are examples of sub information design domains. As indicated, also the information design domain has to do with the enterprise *construction*.

Technology. Appreciably, technology is essential for organizational and informational support, hence an import aspect of the enterprise *construction*. Within the scope of this paper we restrict ourselves to information technology. This design domain is concerned with sub domains like the IT network, application, storage, data communication, etc.

In view of the above, the enterprise construction is defined by three aspect systems: organization, information, and IT, hence three main construction design domains. Together with the business functional design domain, we have the four main enterprise design domains that are schematically depicted in figure 2. The arrows between the design domains express mutual relationships that must be addressed for establishing a unified and integrated enterprise. For all main enterprise design domains (*D*) (and their sub design domains) architecture must be defined that (1) adequately serves as design guidance for ensuring unity and integration, and (2) addresses one or more areas of concern (*A*). Ample examples are provided in [Hoogervorst 2009]. Restricting ourselves to the four main design domains, four main architectures can be identified:

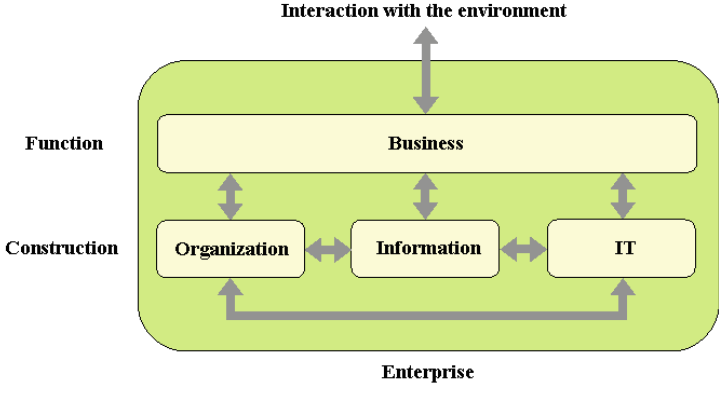


Fig. 2. Main enterprise design domains

Business architecture: the function architecture of the enterprise. The business architecture can formally be defined as a coherent and consistent set of principles and standards that guide the design of the enterprise's function. We might consider principles concerning the provisioning of products and services to customers, the market position relative to competitors or the relationship with stakeholders.

Organization architecture: an aspect of the enterprise construction architecture. The organization architecture is defined as a coherent and consistent set of principles and standards that guide the design of the enterprise's organization for providing the enterprise products and services. Organization architecture for example concerns the design of processes, but also the design of accounting, purchasing, payment, or employee remuneration and evaluation systems. This design perspective is thus necessarily broad. So, establishing desired employee and management behavior, or certain norms and values – all have to do with organizational design.

Information architecture: viewed as a coherent and consistent set of principles and standards that guide the design of an enterprise's information systems. Principles and standards, for example, concern the structure and quality of information, the management of information (gathering, storage, distribution), and the utilization of information. Information architecture is also a sub-architecture of the enterprise construction architecture. Notably, information architecture differs from IT architecture because it is technology independent.

IT architecture: the third sub-architecture of the enterprise construction architecture, defined as a coherent and consistent set of principles and standards that guide the design of IT systems. Principles and standards in this sub-architecture concern all technology dependent IT aspects like network, application, storage, middleware, and so on.

Notably, more detail is required to carry out design activities within the main design domains. More specific sub design domains thus need to be considered, like the sub design domain 'processes' within the main design domain 'organization', as mentioned before. So, process architecture can be seen as a sub-architecture of organization architecture. As indicated before, this points to a hierarchy of design domains and associated architectures: the enterprise at large, the four main design domains, and within these main domains further sub design domains. The latter we have discussed elsewhere [Hoogervorst 2009]. The totality of these architectures is enterprise architecture, which is defined as a coherent and consistent set of principles and standards for the design of the enterprise as a whole. Within the sub design domains, the function/construction perspective likewise holds for the relevant subsystems.

3 Assessing TOGAF

In this section, we will assess TOGAF on the theoretical basis as presented in Section 2. All aspects of TOGAF will be addressed except governance. In our view, governance is not part of an architecture framework or a development methodology. We view enterprise (IT) governance as an organizational competence for continuously exercising guiding authority over enterprise (IT) strategy and architecture development and the subsequent design and implementation of the enterprise (IT system).

This organizational competence must be established for successfully applying the enterprise (IT) development methodology. Moreover, addressing governance in depth would exceed the limited space of this article. From the theoretical basis as presented in Section 2 we derive the seven concrete assessment criteria listed below. After the formulation of each criterion (in italics) we discuss the extent to which it is satisfied by TOGAF.

In order to successfully operationalize strategic changes and ensure enterprise performance, an enterprise must operate as a unified and integrated whole. Unity and integration are not achieved 'incidentally' but must be intentionally created through enterprise design.

Integrated and unified enterprise design seems to be the only way towards mastering the 'organized complexity' of an enterprise [Weinberg 2001]. Assessing TOGAF on this criterion is complicated by the fact that it remains unclear what the objective of TOGAF precisely is. Put differently, TOGAF's notion of enterprise architecture is ambiguous. The descriptive view on architecture (meaning (1) in [The Open Group 2009a, p.9]) is predominant, however. Indicative of this conclusion is the frequent use of terms like "architecture design" and "architecture description". Also 'The Open Group Business Architect Certification Program' discusses architecture in the descriptive sense: in terms of the result of design activities [The Open Group 2009b]. Thus we consider the TOGAF notion of enterprise architecture as some form of a design. Hence, by applying TOGAF, unified and integrated enterprise design can be established, in principle. However, no formal theory and associated methodology is offered for addressing the organized complexity of enterprises and for guaranteeing unified and integrated enterprise design. Although TOGAF purports to be holistic [The Open Group 2009a, p.43] it remains unclear how the holistic view is effectuated. Moreover, effectively establishing enterprise unity and integration requires a scope far exceeding mere IT and process integration [Peters and Waterman 1982, Scott Morton 1991, Doz and Thanheiser 1993, Miles et al. 1995, Martin 1995, Hoogervorst 1998, Daft 2001].

The essential nature of enterprises is their being social systems. Enterprises consist of human beings (actors) who enter into and comply with commitments. In doing so, they collectively realize the enterprise function, and determine enterprise performance.

Given the technology-focused origin of TOGAF, we feel the essential nature of enterprises is not captured by merely extending the IT focus to the 'business domain' in view of the fact that IT supports business processes. Human beings constitute the core of enterprise success and this important aspect appears not to be addressed by TOGAF. As Drucker has pointed out, social aspects of organizing should be the central focus of organizational and management science [Drucker 1985]. Ultimately, enterprise performance is determined by the enterprise's social system [McGregor 1960, Hoogervorst 1998]. This essential aspect must be acknowledged and addressed in any enterprise design methodology.

An important first step in designing an enterprise is capturing its implementation independent essence, also known as enterprise ontology. The ontological model of an enterprise comprises the identified transactions and the corresponding actor roles (chunks of authority and responsibility).

The notion of an implementation independent (ontological) model is totally absent in TOGAF. At the same time, in order to master the organized complexity of enterprises, one needs a perspective that allows one to initially reduce the complexity of enterprises substantially by focusing on the implementation independent essence of the enterprise, i.e. on the enterprise ontology. In doing so, the previously mentioned notion that enterprises are social systems in which human beings enter into, and comply with, commitments is formally operationalized.

Two distinct perspectives are paramount in any design-oriented study of enterprises: function and construction. The function perspective concerns the function and behavior of the enterprise vis-à-vis its environment. The construction perspective concerns the construction and operation of the enterprise.

TOGAF does not seem to offer these perspectives because it lacks an appropriate system notion. Consequently, the system kind remains unclear. As indicated earlier, we consider the TOGAF notion of enterprise architecture as some form of a design. This suggests the enterprise as the system kind of observation. Nonetheless, this perspective is not thoroughly operationalized through functional and constructional enterprise design. It might however very well be that TOGAF's notion of enterprise architecture is not about comprehensive enterprise-wide design. Instead, given the origin of TOGAF, the focus might be on the design of the enterprise 'IT system', which we consider to be the totality of IT subsystems for delivering IT services to the enterprise. But then, the construction perspective on the enterprise as the using system becomes imperative in order to meaningfully determine the IT systems' functions. Only within this perspective the issue of 'business and IT alignment' can be addressed effectively. That brings us back to the enterprise-wide design perspective, which, as pointed out before, seems not to be fully explored by TOGAF.

Enterprise design comprises both function design and construction design. The main enterprise design domains are: business (function), and organization, information, and IT (construction). Through subsequent design, the implementation independent enterprise ontological models are complemented by a sequence of ever more detailed construction models of which the 'lowest' one is implementable.

Creating enterprise unity and integration necessitates comprehensive enterprise-wide functional and constructional design. Despite TOGAF's frequent reference to the notions of 'enterprise' and 'design' no approach for enterprise-wide functional and constructional design is provided. Comprehensive design domains that cover the enterprise in its totality are not indicated. The multiple aspects, as known from the traditional organizational literature, that determine enterprise performance, are thus not or inadequately addressed.

Design pertinent to these domains is respectively guided by business architecture, organization architecture, information architecture, and IT architecture. Collectively they constitute enterprise architecture, where architecture is conceptually defined as the normative restriction of design freedom, and practically as a coherent and consistent set of standards and design principles.

Comprehensive enterprise-wide design entails design guidance, such that unity and integration is achieved and areas of concern are adequately addressed. TOGAF appears to use a vague and ambiguous notion of architecture. Based on the actual usage of the term "architecture", this concept must in fact be understood as a result of design activities, rather than a concept providing design guidance. TOGAF's section on architecture principles rightly so addresses their importance. However, the focus seems to be limited to IT: "Architecture principles define the underlying general rules and guidelines for the use and deployment of all IT resources and assets across the enterprise" [The Open Group 2009a, p.266]. The general, not design-specific character is reflected in the examples given [ibid.]. Enterprise-wide design guidance in the form of normative, prescriptive architecture is arguably not addressed as a central point of attention.

Even if we take into account TOGAF's focus on the enterprise IT system, a comprehensive set of IT system (functional and constructional) design domains must be defined. For these domains architectures must be devised – collectively referred to as IT architecture – that addresses areas of concern and ensure a unified and integrated enterprise IT system design. Pertinent to the enterprise IT system, TOGAF speaks about data, application, and technology architecture. However, not in the normative, prescriptive sense. According to TOGAF, application architecture "provides a blueprint for the individual application systems to be deployed, their interactions, and their relationships to the core business processes of the organization" [op. cit. p.10]. This viewpoint shows TOGAF's lack of a formal distinction between system kinds (cf. Section 2.2) since business processes cannot be addressed within the IT system perspective, but only within the enterprise-wide system perspective. By the same reasoning, the functional IT design domains cannot be addressed within the IT system scope but only within the white-box scope of the enterprise as the using system.

Indeed, TOGAF does introduce the notion of 'business architecture', but in an inappropriate way, as we have seen.

Devising an architecture (called architecturing) must be performed within an architecture framework. Consequently, it regards particular system kinds, design domains, and areas of concern.

As outlined in Section 2, devising an architecture must take place with reference to (1) the system kind *S* of observation, and for which architecture provides design guidance, (2) the design domains *D* were the architecture is used, and (3) the areas of concern *A* that the architecture addresses. We feel that TOGAF lacks a formal distinction between architecturing (devising design principles and standards) and designing, as well as between designing and implementing. TOGAF refers to design activities in phases that should concern implementation only. Implementation and project planning aspects are obviously important topics but, in our view, they should be clearly distinguished from architecturing and designing, hence, should not be part of an architecture framework.

4 Conclusions

Enterprises are (1) goal directed, (2) purposefully designed social systems, that (3) interact with their environments, and for which (4) their unified and integrated operation is conditional for enterprise performance and the successful operationalization of strategic intentions. These four aspects must be central to any design approach for enterprises. Over the years, The Open Group has contributed considerable efforts to devising approaches – collectively identified under the TOGAF label – for enterprise and/or IT system design.. Although these efforts can obviously be appreciated, our analysis forces us to conclude that the TOGAF approach is not grounded in formal, theory-based concepts such that the essential aspects of enterprises are effectively and comprehensively addressed. Moreover, the concepts used are largely ambiguous and ill-defined, thereby making it difficult to master the organized complexity of enterprises. We fail to see how conceptual incoherence and inconsistency can methodically bring forward a coherent and consistent whole. Obvious improvement areas are the clarity, coherence and consistency of the concepts used, and addressing the perspective outlined in section 3.

A major objection one could have is that we have applied our own evaluation framework for assessing TOGAF. To our defense, we can only argue that we have not found another framework that suits our objectives. An often suggested candidate is the GERAM framework [GERAM 1999], since it is intended to facilitate the unification of methods for enterprise integration. However, GERAM offers no theory and associated methodology to accomplish the task. Admittedly, TOGAF acknowledges the importance of unity and integration for proper enterprise and IT systems performance, but it fails to make this operational. Moreover, elements of our framework have been advocated by others too, as referenced in this article.

Arguably, society is largely a society of enterprises. Put differently, societal well-being is to a considerable extent determined by the performance and conduct of enterprises. Adequate performance and conduct (or the opposite) thus have far reaching consequences. For such adequacy, proper enterprise design is crucial. TOGAF aims to address this issue, but fails to do so. As said before, the major cause of this is the lack of a coherent underlying theory. We cannot continue to deal with the immense problems we are facing in an unprofessional way. It is time to practice a profession that is built on sound theoretical foundations, along the lines we have sketched in this paper. Hopefully, our reflection contributes to raise awareness of TOGAF's shortcomings and to rebuild it on solid grounds. This is vitally important in view of the crucial role of enterprises for customers, employees, stakeholders, and the society at large.

References

Ackoff, R.L.: Ackoff's Best: His Classic Writings on Management. Wiley, New York (1999)

Albani, A., Dietz, J.L.G.: Enterprise Ontology based Development of Information Systems. International Journal for Internet and Enterprise Management 7(1) (2010)

Albani, A., Terlouw, L.: An Enterprise Ontology-Based Approach to Service Specification. IEEE Transactions on Services Computing (October-December 2010)

Aveiro, D., Rito Silva, A., Tribolet, J.: Extending the design and engineering methodology for organizations with the generation operationalization and discontinuation organization. In: Winter, R., Zhao, J.L., Aier, S. (eds.) DESRIST 2010. LNCS, vol. 6105, pp. 226–241. Springer, Heidelberg (2010)

Aveiro, D., Rito Silva, A., Tribolet, J.: Towards a G.O.D. Organization for organizational self-awareness. In: Albani, A., Dietz, J.L.G. (eds.) CIAO! 2010. Lecture Notes in Business Information Processing, vol. 49, pp. 16–30. Springer, Heidelberg (2010)

Barjis, J., Kolfschoten, G.L., Verbraeck, A.: Collaborative enterprise modeling. In: Proper, E., Harmsen, F., Dietz, J.L.G. (eds.) PRET 2009. Lecture Notes in Business Information Processing, vol. 28, pp. 50–62. Springer, Heidelberg (2009)

von Bertalanffy, L.: General Systems Theory. George Braziller, New York (1969)

Bryan, L.L., Joyce, C.I.: Better strategy through organizational design. McKinsey Quarterly 2 (2007)

Bunge, M.: Treatise on Basic Philosophy Volume 4: A World of Systems. In: Dordrecht, D. (ed.). Reidel Publishing Company (1979)

Daft, R.L.: Organization Theory and Design. South-Western Publishing, Mason (2001)

Dietz, J.L.G., Albani, A.: Basic notions regarding business processes and supporting information systems. Requirements Engineering 10(3), 175–183 (2005)

Dietz, J.L.G.: Enterprise Ontology. Springer, Berlin (2006)

Dietz, J.L.G.: Architecture: Building Strategy into Design. SDU Publishing, The Hague (2009)

Doucet, G., Gøtze, J., Saha, P., Bernard, S.: Coherence Management: Architecting the Enterprise for Alignment, Agility and Assurance. AuthorHouse, Bloomongton (2009)

Doz, Y., Thanheiser, H.: Regaining Competitiveness: A Process of Organizational Renewal. In: Hendry, J., Johnson, G., Newton, J. (eds.) Strategic Thinking: Leadership and the Management of Change. Wiley, Chichester (1993)

Drucker, P.: Management. Harper, New York (1985)

GERAM, Generalized Enterprise Reference Architecture and Methodology, Version 1.6.3, IFIP-IFAC Taskforce (1999)

Gharajedaghi, J.: Systems Thinking. Butterworth Heinemann, Boston (1999)

Hoogervorst, J.A.P.: Quality and Customer Oriented Behavior: Towards a Coherent Approach for Improvement, Delft, Eburon (1998)

Hoogervorst, J.A.P.: Enterprise Governance and Enterprise Engineering. Springer, Berlin (2009)

Jackson, M.C.: Systems Thinking. Wiley, Chichester (2003)

Jong, J., de, D.J.L.G.: Understanding the realization of organizations. In: Albani, A., Dietz, J.L.G. (eds.) CIAO! 2010. Lecture Notes in Business Information Processing, vol. 49, pp. 31–49. Springer, Heidelberg (2010)

Kaplan, R.S., Norton, D.P.: Norton, Strategy Maps. Harvard Business School Press, Boston (2004)

Kotter, J.P.: Leading Change: Why Transformation Efforts Fail. Harvard Business Review 71(2), 59–67 (1995)

Lawrence, P., Lorsch, J.: Organization and Environment. Harvard Business School Press, Boston (1967)

Leinwand, P., Mainardi, C.: The Coherence Premium. Harvard Business Review (June 2010)

Luthans, F.: Organizational Behavior. McGraw-Hill, New York (1992)

Maier, M.W., Rechtin, E.: The Art of Systems Architecting. CRC Press, Boca Renton (2002)

Martin, J.: The Great Transition. In: Using the Seven Principles of Enterprise Engineering to Align People, Technology and, Strategy. American Management Association, New York (1995)

McGregor, D.M.: The Human Side of Enterprise. McGraw-Hill, New York (1960)
Miles, R.E., Coleman, H.J., Douglas Creed, W.E.: Success in Corporate Redesign. California Management Review 37(3), 128–145 (1995)
Mintzberg, H.: The Rise and Fall of Strategic Planning. The Free Press, New York (1994)
Nadler, D.A., Tushman, M.L.: Competing by Design: The Power of Organizational Architecture. Oxford University Press, New York (1997)
van Nuffel, D., Mulder, H., van Kervel, S.: Enhancing the formal foundations of BPMN by enterprise ontology. In: Albani, A., Barjis, J., Dietz, J.L.G., et al. (eds.) CIAO! 2009. Lecture Notes in Business Information Processing, vol. 34, pp. 115–129. Springer, Heidelberg (2009)
Peters, T.J., Waterman, R.H.: In Search of Excellence. Warner Books, New York (1982)
Pettigrew, A.: Success and Failure in Corporate Transformation Initiatives. In: Galliers, R.D., Baets, W.R.J. (eds.) Information Technology and Organizational Transformation. Wiley, Chichester (1998)
Rechtin, E.: Systems Architecting of Organizations. CRC Press, Boca Raton (2000)
Scott Morton, M.S.: The Corporation of the 1990s. Information Technology and Organizational transformation. Oxford University Press, New York (1991)
Schein, E.H.: Organizational Culture and Leadership. Wiley, Chichester (2004)
The Open Group, TOGAF Version 9. Van Haren Publishing, Zaltbommel (2009a)
The Open Group, The Open Group Business Architect Certification Program, San Francisco (2009b)
Terlouw, L., Dietz, J.L.G.: A Framework for Clarifying Service-Oriented Notions – How to Position Different Approaches. Enterprise Modeling and Information Systems Architectures 5(1) (July 2010)
Ven, K., Verelst, J.: The adoption of DEMO: A research agenda. In: Albani, A., Barjis, J., Dietz, J.L.G. (eds.) CIAO! 2009. Lecture Notes in Business Information Processing, vol. 34, pp. 157–171. Springer, Heidelberg (2009)
Weinberg, G.M.: An Introduction to General Systems Thinking. Dorset House Pub-lishing, New York (2001)
Winograd, T.: A Language/Action Perspective on the Design of Cooperative Work. Human-Computer Interaction 3(1), 3–20 (1988)
Winograd, T., Flores, F.: Understanding Computers and Cognition: A New Foundation for Design. Addison-Wesley, Boston (1987)

A Method to Develop EA Modeling Languages Using Practice-Proven Solutions

Sabine Buckl, Florian Matthes, and Christian M. Schweda

Chair for Software Engineering of Business Information Systems (sebis),
Technische Universität München,
Boltzmannstr. 3, 85748 Garching, Germany
{sabine.buckl,matthes,christian.m.schweda}@mytum.de
http://wwwmatthes.in.tum.de

Abstract. Enterprises are unique in the way of doing business. This uniqueness is typically reflected in the overall make up of the enterprise – the enterprise architecture (EA). Globalized markets, changing legal regulations, and technological innovations thereby force enterprises to continually adapt their EA to the changing environment. As response, enterprises aim at a strategic management of the EA providing a holistic model of the key elements and relationships of an enterprise. Different supporting modeling languages have been proposed but none of them has gained broad acceptance due to the above described uniqueness.

In this paper we present a method to develop organization-specific EA modeling languages based on building blocks representing practice-proven solutions. Following the common understanding of modeling languages as consisting of syntax, semantics, and notation, we provide three different types of building blocks: information model building blocks that specify the syntax, glossary building blocks that textually define semantics, and viewpoint building blocks that specify the notation of the language. The applicability of the method for integrating building blocks to a consistent EA modeling language is illustrated along a case study from the public sector. The exposition of the method concludes with an outlook on further areas of research.

1 Motivation and Overview

Enterprise architecture (EA) management is a discipline, which has recently gained increased attention from academia and practice. Thereby, a few topics which are nowadays regarded to be part of EA management, have a long history in information systems research. This can be exemplified with the topic of business-IT-alignment discussed e.g. by Henderson and Venkatraman in the late nineties as strategic alignment [1]. While these discussions might have catalyzed the evolution of EA management, the overall discipline is still subject to ongoing development. This in particular applies as different research communities continue to argue on the perspective, from which EA management should be approached (cf. discussions by Frank [2] or Wegman [3]). The approaches

A. Albani, J.L.G. Dietz, and J. Verelst (Eds.): EEWC 2011, LNBIP 79, pp. 91–105, 2011.

nevertheless agree that EA management needs to provide a holistic view on an enterprise, accounting for aspects from all layers, ranging from business to IT.

Independent from the question of perspective, other indications for the ongoing development of the EA management discipline exist. EA modeling represents a prominent example. Although most EA management approaches emphasize on the importance of modeling the EA, no common metamodel, called **information model** in accordance with Buckl et al. in [4], has yet been established. In the last years, many information models were proposed but none of them has gained broad acceptance. Some researchers even challenge the hypothesis that such a model exists (cf. [5,6]). They expect enterprises to have largely different expectations on the benefits of EA management, and therefore assume that an information model is an enterprise-specific artifact. In Section 2 we discuss possible reasons for this kind of specificity, when we revisit how today's EA modeling languages position themselves as tools for supporting EA management functions. In [5] Buckl et al. discuss three different ways for developing organization-specific EA modeling languages, of which two approaches, namely the *customization approach* and the *integration approach* try to leverage established best-practices. Understanding that most of today's EA and EA management frameworks pursue a customization-based approach, we discuss the shortcomings of such approach which motivates the research objective of this article:

> Introduce a method for developing EA information models based on composeable best-practice solutions for defining such languages, and show the applicability of the method.

This objective is approached in Section 3, where we outline our method for developing organization-specific EA modeling languages, which is based on reusable **building block**s for such languages. The method is based on a specific understanding of EA modeling languages, according to which the language's syntax is specified in an information model, the notation is defined via **notation function**s as well as **representation function**s, and the semantics is defined textually in a **glossary**. For any of these constituents, the presented method supplies specific building blocks, which are further composed into a comprehensive EA modeling language. The applicability of the method is exemplified in Section 4. Final Section 5 summarizes the article's findings and gives an outlook on further research to follow in the field.

2 EA Modeling: Theoretic Foundations and State-of-the-Art

A plethora of different EA management approaches has been proposed in the past that typically either focus on a language to model the EA or on a method how to document, analyze, and communicate EA-related aspects. This section provides an overview on selected EA management approaches with particular focus on the proposed EA modeling language(s). Preparing the analysis of existing EA modeling languages, we establish a theoretic foundations and revisit a

conceptual framework for EA design as discussed by Buckl et al. in [7]. Central to this framework is the understanding that EA modeling has both an intensional and an extensional nature. Extensionally, languages are used to reflect certain architectural properties, i.e. phenomena, whereas the intensional nature of a language reflects the fact that the corresponding models are created for 'doing something'. Understanding EA management as a design process, i.e. as engineering process targeting the transformation of the enterprise, Buckl et al. apply in [7] the propositional framework for design of Simon [8]. This framework promotes a formal understanding of the activity of design, based on the central notion of *means-end*-relationships. Any design activity is carried out in the light of domain-inherent characteristics and relationships ("natural laws" [8]) that connect dedicated means to corresponding ends. This conversely means that a designer with a specific end (goal) in mind searches for means by which the design artifact will achieve those ends. For the context of EA management, this means that enterprise architects "given the constraints and fixed parameters, find values for the command variables that satisfy the utility function" [8]. Building on this, Buckl et al. explored in [7] how the constituents of EA-specific design propositions look alike and devised a mapping distinguishing between:

- the **current** and **future** make-up of the EA (command variables),
- the **strategies & projects** affecting and changing the EA (means),
- the **principles & standards** guiding the evolution of the EA (constraints),
- the **visions & goals** describing a target state of the EA (ends), and
- the **KPIs & metrics** measuring and evaluating an EA state (utility function).

Above distinction pertains to the EA modeling language, more precisely to the language constituent of the syntax that introduces the primitives used in an EA model. In line with Ernst et al. [9], we assume that the language syntax is represented in an **EA information model**, containing the classes, properties, and associations used to describe a particular **area-of-interest** in the EA. These areas-of-interest can be distinguished to extensional ones (**concerns**), i.e. ones covering the command variables, and intensional ones, covering **cross-cutting** aspects, as strategies, goals, standards, or metrics. This distinction is to be kept in mind with respect to the review of the state-of-the-art undertaken in this section, but also for devising our development method in Section 3. The distinction gives rise to a particular conception of the EA modeling language(s) used to support a particular EA management approach.

TOGAF proposes to structure the EA in different architecture domains representing subsets of the overall EA [10]. TOGAF distinguishes between

- *business architecture* concerned with strategic, governmental, organizational, and process-related aspects,
- *data architecture* describing the structure of an organization's data assets and data management resources,
- *application architecture* considering the application systems, their interactions, and their relationships to the business processes, and

– *technology architecture* describing the software and hardware capabilities required to support business, data, and application services.

Answering the question which elements should be considered in an EA management endeavor, TOGAF presents the *core content metamodel*. Therein, the entities and relationships that make up an EA are described [10]. The core content metamodel provides entities for all architectural layers. Besides the entities, which can be grouped to one of the architectural layers, TOGAF also introduces crosscutting entities associated with all objects among others principle, requirement, work package. Thereby, the kind of relationship is not discussed. TOGAF provides six metamodel extensions [10], e.g the *motivation extension* to enable measurement of business performance by introducing concepts as driver, goal, and objective. While each of these extensions supplies concepts for modeling and described the intended usage context, these concepts are not formulated as cross-cutting aspects pertaining to arbitrary architecture elements.

In the 1970s and the 1980s several EA-related frameworks have been developed. In response to the emerging number of frameworks in this area, the *International Task Force on Enterprise Integration* was established aiming at the development of a reference framework that supports comparison and evaluation of existing approaches [11]. As a result of the investigation, the Task Force developed the *Generalised Enterprise Reference Architecture and Methodology* (GERAM). GERAM consists of nine components, of which with respect to the EA modeling language, three components are of interest. These do not impose particular languages but define criteria for an EA management approach [12]:

– *Generalised Enterprise Reference Architecture* (GERA): GERA describes the basic concepts to be used in enterprise engineering and integration projects. According to GERAM these concepts can be categorized as human-oriented concepts, process-oriented concepts, and technology-oriented concepts.
– *Enterprise Modeling Languages* (EMLs): EMLs define the generic modeling constructs for enterprise modeling. In particular, the EMLs provide constructs to describe and model human roles, operational processes, supporting information, and technologies.
– *Generic Enterprise Modeling Concepts* (GEMCs): GEMCs define and formalize the generic concepts of enterprise modeling. The following ways of formalization exists: natural language explanations (*glossaries*), meta models describing the elements and their relationships (*information models*), and theories defining the semantics of enterprise modeling languages (*ontologies*).

GERA defines a life-cycle for each constituting concept of the enterprise, which consists of the phases *identification*, *concept*, *requirements*, *(preliminary and detailed) design*, *implementation*, *operation*, and *decommission*. While most of the aforementioned phases are self-explanatory, the *concept* phase deserves a more in depth analysis with respect to our analysis framework. The phase is concerned with the definition of the entity's mission, vision, strategies, objectives, etc. [12]. Thus, linking the cross-cutting aspects of strategies, projects, visions, and goals

to any concept considered during enterprise transformation. In line with the objective of GERAM to define requirements for EA (management) frameworks, no description how this relation should be conceptualized is given. Similarly, the concept of *life history* is discussed and the link to different kind of projects is explored and related to the phases of the EA concepts.

In addition, to the generalized propositions for a language for EA descriptions as discussed above, the EMLs define two requirements to enable integration of special purpose modeling languages (cf. [12]). First, every area as represented in the modeling framework must be covered for every enterprise entity type, and second, any model developed must be able to be integrated with models of other subject areas, if the information content of the model requires integration. The need to integrate different languages results from the distinct 'expressive powers' related to the intended purpose, e.g. description vs. analysis, of the languages.

Against the background of over 15 years of practice, Dietz [13] has developed a "methodology for (re)designing and (re)engineering organizations" called DEMO. With its sound theoretical foundation in a theory called Ψ-theory the method takes a different perspective on the enterprise focusing on the so-called "enterprise ontology". Dietz uses this term to denote a "coherent, comprehensive, consistent and concise model of the essence of the enterprise". Critical to his definition is thereby the notion of "essence" that in the sense of Dietz targets the deep behavioral nature of the enterprise, but not realization and implementation specific details. The method of DEMO provides an approach to develop enterprise ontologies in a systematic way [13], i.e. reflects commitment-related information. In line with the four basic axioms of Ψ-theory the ontological model of the enterprise is constituted of four distinct submodels (*construction model*, *state model*, *process model*, and *action model*) The *construction model* specifies the construction of the organization embodied in the identified transaction types as well as actor roles. Detailing the coordination aspect of the transactions, the *process model* describes causal and conditional relationships between different transactions. Complementing this perspective the *state model* outlines the state space of P-facts, i.e. of production results, while P-acts are not part of the state model, as they may be derived from the corresponding process model. The *action model* describes the enterprise ontology on the most detailed model, such that – as Dietz states in [13] – the other models may be derived from the action model, and are hence only provided for ease of use. The different abstracted models (construction, process and state model) are complemented each with a specific description language, of which especially the language behind the state model deserves special attention. The so called "world ontology specification language" (WOSL) (cf. Dietz [14]) provides the basic language elements for describing *rigid* and *non-rigid* structures, i.e. states that exist universally over time and states that may change. The construction and process model languages present themselves as two sides of the same coin taking a blackbox and a whitebox perspective on the organizational transactions further mirrored in the prescriptive understanding of an EA complementing the enterprise ontology with "functional" and "constructional principles" [13].

3 Designing an EA Modeling Language

Different EA management problems call for a distinct understanding of the EA and hence entail a different way of modeling architectural properties. For each such problem, the corresponding understanding of the problem domain, i.e. the relevant part of the EA, can be expressed in a specific EA modeling language. This language is *domain appropriate* in terms of Krogstie [15], i.e. constrains itself only to relevant syntax, semantics, and notation. For developing one or more EA modeling languages, e.g. representing different problems, we have to design the corresponding syntax, semantics, and notation. Prior to introducing our design method, we provide a conceptual model that describes the interplay of these constituents. Central thereto is the **information model**, which in line with our considerations in [9], specifies the syntax of the modeling language via MODEL ELEMENTS. Semantics is considered as a function assigning exactly one **predicator** [16] to each MODEL ELEMENT. This predicator acts as surrogate for a corresponding part of the conceptualization, i.e. for an *architecture property* in the sense of Dijkman et al. [17]. Such property represents a characteristic of the architecture relevant in respect to one or more architecture stakeholders. Complementing the notation is described as a representation function assigning one VISUALIZATION ELEMENT[1] to each MODEL ELEMENT. In consequence, a predicator (or the thereby represented architectural property) is assigned a particular visualization element by the modeling language.

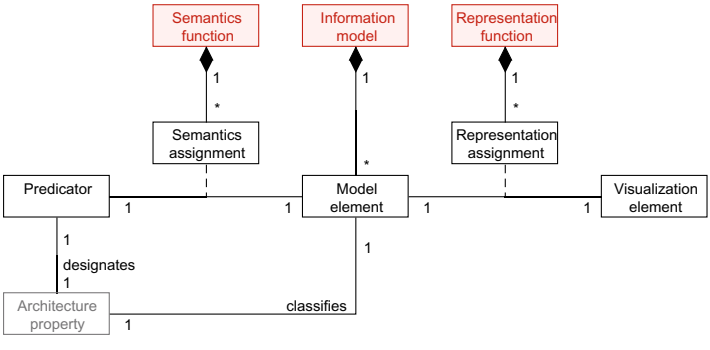


Fig. 1. General constituents of an EA modeling language

Different languages in turn can cover different sets of predicators and can assign different visualization elements thereto. Figure 1 depicts the conceptual model for modeling languages utilizing aforementioned terms. The architecture property is therein shown in gray, as it represents a purely intrapersonal concept, whereas the three basic constituents of the language syntax (INFORMATION

[1] In line with Kamlah and Lorenzen [16] arbitrary *speech acts* can notate an element. With the focus of the domain, we constrain our subsequent considerations to visual elements.

MODEL), semantics (SEMANTICS FUNCTION), and notation (REPRESENTATION FUNCTION) are highlighted.

Combining different modeling languages in a consistent manner builds on the notion of consistency as introduced by Dijkman in [18]. In particular, the question of integrating the underlying information models is of relevance, as these can contain different MODEL ELEMENTs reflecting the same architecture property. Such MODEL ELEMENTs originate from different ways of perceiving the underlying property. For the context of the method, the predicator, i.e. the glossary entry, identifies the corresponding property. Two information models can be interlinked by three different types of relationships, namely **overlap**, **subsume**, and **conflict**. Moving from the instance-level perspective on relationships as taken by Dijkman in [18] to an understanding on a conceptual level, we use the predicators and define the basic relationship overlap as expressing that two information models share at least one predicator. Building on this, the two other types of relationships are defined as follows:

- Two information models conflict with each other, if they overlap, but make contradictory statements with respect to the MODEL ELEMENTs assigned to overlapping predicators.
- One information model subsumes another one, if the subsuming model is completely overlapped by the subsumed one, i.e. if all predicators of the subsuming information model are also present in its subsumed counterpart.

Subsume and **conflict** are closely related to each other in the sense that for two conflicting information models it is not possible to find or create a third information model subsuming both. Contrariwise, a subsumed information model completely overlaps the subsuming one without raising conflicts. The types of relationships between information models provide a basis for our development method in a threefold manner. Firstly, the users can be prevented from selecting conflicting information models to be integrated into the comprehensive information model supporting their EA management function. Secondly, existing overlaps between the selected information models can be accounted for during integration of the information models. Finally, the subsume-relationship can be used to derive possible paths for evolving the information model.

Linking information models to visualizations, we have to define two types of functions, namely the **notation function** and the **representation**

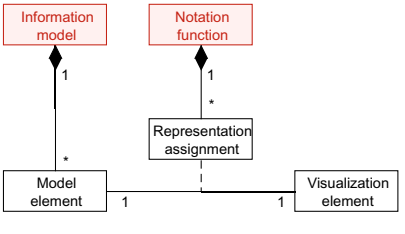


Fig. 2. Notation function

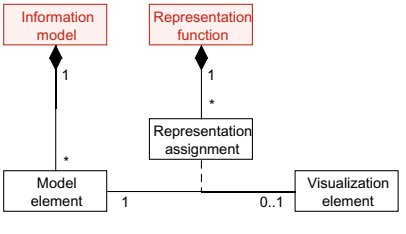


Fig. 3. Representation function

function. For each modeling language the notation function establishes a bijective mapping between MODEL ELEMENTs and VISUALIZATION ELEMENTs. The representation function of other modeling languages contrariwise does not establish one-to-one relationships, but can provide an aggregated perspective on the information model elements, e.g. by abstracting relationships or calculating sums. Put in terms of our conceptual model for the constituents of an EA modeling language, we can describe the general distinction between notation functions and representation functions via the existence of representation assignments as shown in Figures 2 and 3, respectively.

The solutions provided by the different approaches to EA management, discussed in Section 2, provide partial EA modeling languages, which are documented on different levels of abstraction, giving several specifics of the application, e.g. the employed terminology. EA management approaches with an embracing appeal tend to mitigate specifics of a singular prescription and to present the solutions in a *stratified* terminology, thereby increasing readability and comparability. In this sense the assignments between the PREDICATORs and the MODEL ELEMENTs as contained in different EA modeling languages of the approach are adapted. Any EA modeling language can supply at most one MODEL ELEMENT assigned to a particular PREDICATOR from the stratified terminology of the approach. Nevertheless, different languages can bring along elements assigned to the same predicator. Figure 4 displays the conceptual model bringing together different EA modeling languages backed in a consistent terminology.

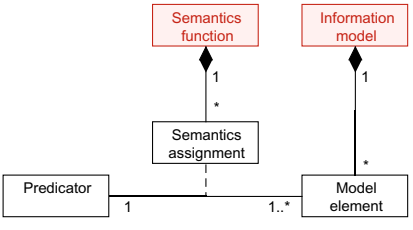


Fig. 4. Semantics assignments in a set of EA modeling languages

Our development method has to target the three aspects of EA modeling languages with corresponding building blocks. These building blocks have been proven to work in practice [19]. In particular *information model building block* (IBB) supplying an information model, *viewpoint building block* (VBB) supplying a representation function, and *glossary building block* (GBB) supplying a semantics function. These building blocks are complemented with techniques facilitating their integration and combination. Using the techniques and building blocks of the different types a consistent set of EA modeling languages can be developed. Figure 5 shows the interplay of the different practice-proven building blocks complementing the development method.

With respect to IBBs, our method offers an additional distinction between MODEL ELEMENTs that reflect architecture elements, representing the

extensional nature, and those that reflect goals of EA management, representing the intentional nature. Revisiting the analysis of EA modeling languages from Section 2, we understand goals as particular instantiation of a more general principle of describing EAs. Similar to goals, also projects or standards raise certain characteristics that do not supply an *identity condition* (IC) for the corresponding model elements. Put in other words, while goals, projects, and standards themselves supply an IC, they relate to dependent model elements that do not supply an IC, are *dispersive* types [20]. Based on this analysis, we establish a distinction between two types of IBBs: one building around identifiable architecture elements (**concern IBB**s) and one centering around cross-cutting architecture characteristics (**cross-cutting IBB**s). These concepts are further summarized under the term **area-of-interest**.

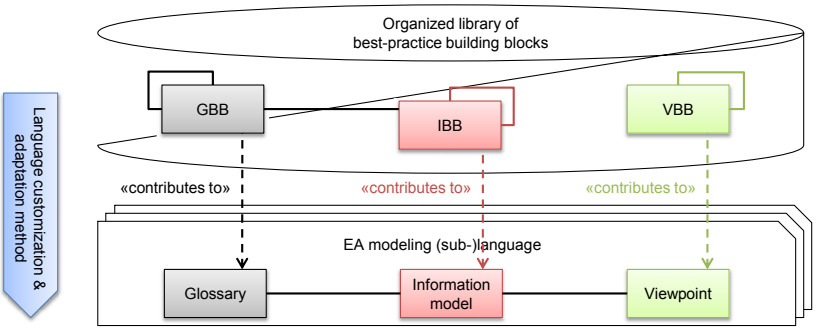


Fig. 5. Interplay of language building blocks

An IBB defines a part of the syntax of an EA modeling language by specifying the corresponding MODEL ELEMENTs, i.e. the TYPEs, RELATIONSHIPs, and PROPERTYs, that make up the syntax. In developing a specific information model, cross-cutting IBBs corresponding to goals and questions, projects, or standards are applied, i.e. added, to concern IBBs. Via supplied semantics assignments, the model elements are linked to predicators, i.e. classifying terms from the glossary, which in turn reflect a particular architecture property. The different values that an architecture property can take are codified into instantiations of the corresponding MODEL ELEMENT. If this concept, for example, is a PROPERTY[2] these instances can be identified with the range of the corresponding data type, e.g. INTEGER or STRING. Similar considerations also apply for TYPEs and RELATIONSHIPs. With the IBBs specifying re-usable information models bound to a consistent underlying terminology, the relationships applying on information models can also be applied to interrelate IBBs. The IBB-relationships are of interest in extending the method's underlying organized collection of IBBs as exemplified in Figure 6. Each of these relationships is therein set effective by at least one predicator.

[2] We use the different typeset to avoid confusion with the architectural property.

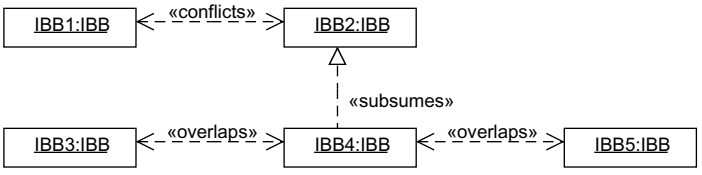
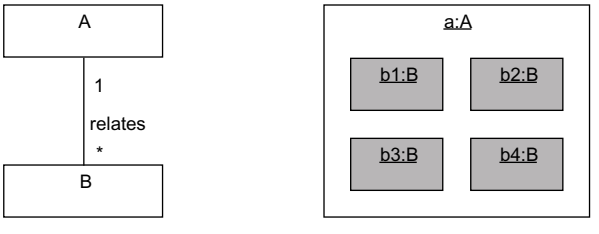


Fig. 6. Exemplary net of IBB-relationships

In order to ensure usability of the method in general and the techniques for selecting and integrating the IBBs in special, any newly added IBB has to be embedded into the net of IBB-relationships, if overlapping. Forcing the contributors of IBBs to establish these relationships manually would aggravate any extension of the organized library of building blocks, exponentially growing with the number of already supplied IBBs. Resolving the aforementioned difficulty, we decide to derive these relationships from the relationships between predicators and model elements as well as the containment of model elements in the IBBs.

Another important aspect of developing EA modeling languages from practice-proven solutions is the notational aspect. Any language brings along symbolic representations for the MODEL ELEMENTs and their relationships, which must in turn match the specific notational expectations of the corresponding users. With respect to the method, this means that the users must be able to select proven-practice notations and adopt them to fit their particular EA modeling languages. The VBBs provide such proven-practice representation assignments as derived from the analyzed EA management approaches. These assignments are of abstract nature, i.e. do not relate to a particular underlying information model, but to an abstract conception of the information to be represented. Latter forms the **viewmodel**[3], which can be understood as a 'virtual' information model, acting as domain for a representation function. Revisiting the exemplary visualization shown in this section, we explain the notion of the viewmodel along the example of the *clustered visualization*, which according to Wittenburg [21] is frequently used in representing EA information[4].



A clustered visualization depicts instances of an outer TYPE A and instances of an inner TYPE B, which are related via some RELATIONSHIP.

[3] A similar concept is discussed by Fowler in http://martinfowler.com/eaaDev/PresentationModel.html (cited 2010-12-23).

[4] Wittenburg calls this type of visualization "cluster map" [21, page 78–79].

The viewmodel supplies enough information to create the visualization but not more, such that the representation assignments constitute a bijective representation function. It would nevertheless be possible to create the visualization in cases, where additional information was supplied. In mathematical terms this means that any particular kind of visualization, i.e. any VBB, can be applied on any information model capable of supplying the **sufficient information**. For being in turn able to perform graphical modeling, the underlying information model must not supply additional information, i.e. has to supply only the minimal **necessary information**. Such information model is closely related to the viewmodel of the corresponding VBB. These considerations take a different point of view on what differentiates notation functions and graphical representation functions. While the former relate visualizations to their necessary and sufficient information model (the viewmodel), latter functions can build equivalently on any sufficient information model. Being sufficient relates to the concept of the subsume-relationship introduced above. Any information model that is subsumed by visualization's necessary and sufficient information model is itself sufficient to create the visualization. Understanding the subsume-relationship as a basis for evolving the information model of an organization, we can ensure that any subsumed information model is still sufficient to create the once selected graphical representations. Aforementioned indications give rise to the following definition of VBBs. In line with our considerations in [22] we detail on the contribution that a VBB makes with respect to specifying a viewpoint. According to the provided contribution, we distinguish different types of VBBs:

- **Symbol VBB**s that assign a mapping between a MODEL ELEMENT and a visible VISUALIZATION ELEMENT, i.e. a symbol in terms of [9].
- **Structural VBB**s that assign a complex structure of MODEL ELEMENTs to a set of interrelated VISUALIZATION ELEMENTs. Such building blocks are mostly not self-contained but reference other VBBs as sub-assignments.
- **Decorating VBB**s that assign a mapping between a MODEL ELEMENT and a VISUAL PROPERTY of a VISUALIZATION ELEMENT, which in turn results from a mapping specified in a different VBB.
- **Hybrid VBB**s that specify to some extent a combination of the above. Such VBBs can be constituted from other VBBs, e.g. a structural VBB referencing an elementary VBB to which additionally a decorating VBB is applied.

Any viewpoint is configured in terms of at least one structural VBB that determines the overall make-up of the corresponding visualization. This understanding aligns with the principle of the *base map* as established by Wittenburg in [21] used to describe the basic nature of the analyzed *software maps*. It is nevertheless not necessary for a viewpoint to build on an isolated structural VBB, as also a hybrid VBB containing at least one structural VBB can serve as **base VBB** for a viewpoint. Finally, the GBBs are used to provide a consistent underlying understanding of the concepts employed in the IBBs. Central thereto is the notion of the predicator, which is complemented with a textual description of the according semantics. This description can in turn link to related predicators,

although the relationships established thereby are not typed, but give indications on a corresponding connectedness.

Complementing above considerations on the building blocks and their interrelationships to each other, we subsequently outline how the organized library of building blocks is used by the development method proposed in this paper. In general building block-based development of EA modeling languages consists of three distinct activities, namely **information modeling**, **viewpoint definition**, and **glossary adaptation**. Latter activity is elemental in its nature, meaning that the users of the method browse through an existing glossary and rename one or more of its entries. Thereby, the corresponding predicator is shadowed with an organization-specific predicator, that consistently inherits all semantic assignments of its underlying predicator. Information modeling starts with an empty information model and iteratively applies the following four steps:

1. Select EA management-relevant goal to pursue and choose cross-cutting IBB reflecting the goal on an adequate level of abstraction.
2. Select EA concern on which the goal applies and choose admissible concern IBB reflecting the concern on an adequate level of abstraction.
3. Integrate the cross-cutting IBB with the concern IBB to build a problem-specific information model by specifying the goal-relevant MODEL ELEMENTS.
4. Integrate the problem-specific information model with the already existing information model. (omitted in first iteration)

After above steps have been executed once, an information model is maintained by the method in the organization-specific configuration. This information model is used to determine, whether a concern IBB is admissible or not. The latter is the case, when the IBB conflicts with an already integrated one. Another subtlety applies to this step, as the concern descriptions change with the adaptation of the glossary, thus ensuring that the users perform the selection based on their adapted terminology. The activity of viewpoint definition is closely interrelated to the one defining the EA management function in general. The method part of the function specifies particular tasks, in which actors in EA management and EA stakeholders have to be informed or must provide information on certain architectural aspects. With the viewpoints, in line with Krogstie [15], being the vehicles for externalizing, comprehending, and communicating information, any relationship between an actor or stakeholder and a task has to be supplied with a specific viewpoint. Thereto, the following two step approach is applied:

1. Select base VBB and establish admissible links from its virtual information model to the model of available information.
2. Repeat: add VBB to detail existing viewpoint and establish admissible links from the corresponding virtual information model to the model of available information.

Two conceptions in the former approach deserve special attention. Firstly, it can be the case that in some tasks of the EA management function not all information according to the integrated information model is available. This particularly

applies for documentation-related tasks, such that any VBB applied to define a viewpoint is confined to the information actually available. Put in other words, the VBB has to be configured against the model of available information. Secondly, while there are in general only a few restrictions on the VBB to apply for a specific task, the configuration of the corresponding viewpoint is restricted with respect to the intended usage thereof. Having selected a task-actor-relationship, according to which the actor has to provide information about an EA concern, the configured viewpoint must supply representation assignments that constitute a notation function for the information under consideration. Put in other words, the actors must have the possibility to model the corresponding part of the enterprise using the established modeling language. In this sense, any mapping configuration is analyzed by means of the provided techniques with respect to its suitability to maintain a bijective relationship with the underlying information model, i.e. if this information model is isomorphic to the VBB's viewmodel.

4 Exemplifying the Design Approach

A public authority providing IT services to several federal ministries has over the years 'grown' a highly heterogeneous application landscape, causing high maintenance costs and requiring a diverse set of IT operating skills. In order to address this IT-related problem, the authority decides to introduce an EA management function specifically pursuing the goal of *standardization* on the level of *business applications* and their underlying *technologies*. Having chosen goal and concern, two IBBs become available, one describing how to operationalize standardization, another defining the concepts needed to describe the relationship between applications and technologies. Figure 7 depicts the information model derived from these IBBs via integration.

The complementing GBBs for the information model elements, introduce definitions. The public authority decides to apply the definitions without adaptations. Finally, the authority designs a viewpoint for communicating the information about standardization. The viewpoint builds on a basic cluster-VBB extended with a VBB for color-coding, according to which non-standard technologies are colored red, whereas standard technologies are colored green.

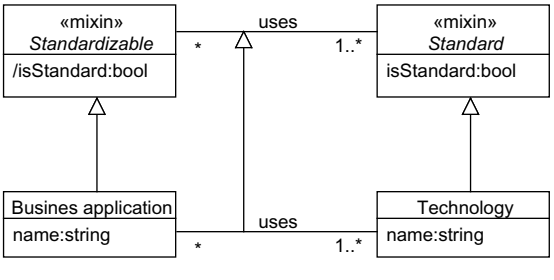


Fig. 7. Information model

5 Conclusion and Outlook

Balancing organization specificity on the one hand and the demand for a general method to model EAs, we proposed in this paper a method to develop organization-specific EA modeling languages based on practice-proven building blocks. To enable organization-specific configuration, three different types of building blocks have been proposed. Information model building blocks describing the syntax of the language, i.e. the concepts that make up the EA, glossary building blocks that specify the semantics of the concepts reflecting the organization-specific terminology, and viewpoint building blocks providing a stakeholder-specific notation for visualizing EA-related information. While the different building blocks enable flexible configuration to an organization-specific EA modeling language, aspects of consistency have to be accounted for. In presenting our development method we discussed how consistency can be ensured.

While the utilization of the development method in a case study with an industry partner from the public sector provides first indications for the applicability and utility of the presented method, further case studies should be conducted. Furthermore, a tool supporting the user of the development method in conducting the single steps of the method can be regarded useful. In particular as such a tool could be used as a configurator for that would enable initial design as well as adaptation of EA modeling languages thus supporting the enterprise architects in adapting to changing environmental influences and problems to be addressed. The resulting configuration could further be used as input for dedicated EA management tools to facilitate the customization thereof.

References

1. Henderson, J.C., Venkatraman, N.: Strategic alignment: leveraging information technology for transforming organizations. IBM Systems Journal 32(1), 472–484 (1993)
2. Frank, U.: Multi-perspective enterprise modeling (memo) – conceptual framework and modeling languages. In: 35th Hawaii International Conference on System Sciences (HICSS 2002), Washington, DC, USA, pp. 1258–1267 (2002)
3. Wegmann, A.: On the systemic enterprise architecture methodology (seam). In: SEAM. Published at the International Conference on Enterprise Information Systems (ICEIS 2003), pp. 483–490 (2003)
4. Buckl, S., Ernst, A.M., Lankes, J., Matthes, F., Schweda, C.M., Wittenburg, A.: Generating visualizations of enterprise architectures using model transformation (extended version). Enterprise Modelling and Information Systems Architectures – An International Journal 2(2), 3–13 (2007)
5. Buckl, S., Ernst, A.M., Lankes, J., Schneider, K., Schweda, C.M.: A pattern based approach for constructing enterprise architecture management information models. In: Oberweis, A., Weinhardt, C., Gimpel, H., Koschmider, A., Pankratius, V., Schnizler (eds.) Wirtschaftsinformatik 2007, pp. 145–162. Universitätsverlag Karlsruhe, Karlsruhe (2007)

6. Kurpjuweit, S., Winter, R.: Viewpoint-based meta model engineering. In: Reichert, M., Strecker, S., Turowski, K. (eds.) 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2007). LNI, Bonn, Germany, Gesellschaft für Informatik, pp. 143–161 (2007)
7. Buckl, S., Matthes, F., Roth, S., Schulz, C., Schweda, C.M.: A conceptual framework for enterprise architecture design. In: Proper, E., Lankhorst, M.M., Schönherr, M., Barjis, J., Overbeek, S. (eds.) TEAR 2010. Lecture Notes in Business Information Processing, vol. 70, pp. 44–56. Springer, Heidelberg (2010)
8. Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)
9. Ernst, A.M., Lankes, J., Schweda, C.M., Wittenburg, A.: Using model transformation for generating visualizations from repository contents – an application to software cartography. Technical report, Chair for Informatics 19 (sebis). Technische Universität München, Munich, Germany (2006)
10. The Open Group: TOGAF "Enterprise Edition" Version 9(2009), http://www.togaf.org cited 2010-02-25
11. Bernus, P., Nemes, L., Schmidt, G.: Handbook on Enterprise Architecture. Springer, Heidelberg (2003)
12. IFIP-IFAC Task Force on Architecture for Enterprise Integration: Geram: The generalised enterprise reference architecture and methodology. In: Bernus, P., Nemes, L., Schmidt, G., eds.: Handbook on Enterprise Architecture, pp. 21–63. Springer, Heidelberg (2003)
13. Dietz, J.L.: Enterprise Ontology. Springer, Heidelberg (2006)
14. Dietz, J.L.: A world ontology specification language. In: On the Move to Meaningful Internet Systems 2005: OTM Workshops, pp. 688–699 (2005)
15. Krogstie, J.: A semiotic approach to quality in requirements specifications. In: Proceedings of the IFIP TC8 / WG8.1 Working Conference on Organizational Semiotics: Evolving a Science of Information Systems, pp. 231–249. Kluwer, Deventer (2002)
16. Kamlah, W., Lorenzen, P.: Logische Propädeutik: Vorschule des vernünftigen Redens, 2nd edn. Metzler, Stuttgart (1967)
17. Dijkman, R.M., Quartel, D.A., van Sinderen, M.J.: Consistency in multi-viewpoint design of enterprise information systems. Information and Software Technology 50(7–8), 737–752 (2008)
18. Dijkman, R.M.: Consistency in multi-viewpoint architectural design. PhD thesis. Enschede (2006)
19. Buckl, S., Matthes, F., Schweda, C.M.: Utilizing patterns in developing design theories. In: 2010 International Conference on Information Systems, ICIS 2010 (2010)
20. Guizzardi, G.: Ontological foundations for structural conceptual models. PhD thesis, CTIT, Centre for Telematics and Information Technology. Enschede, The Netherlands (2005)
21. Wittenburg, A.: Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften. PhD thesis, Fakultät für Informatik, Technische Universität München, Germany (2007)
22. Buckl, S., Gulden, J., Schweda, C.M.: Supporting ad hoc analyses on enterprise models. In: 4th International Workshop on Enterprise Modelling and Information Systems Architectures (2010)

Modularity in Enterprise Architecture Projects: An Exploratory Case Study

Philip Huysmans, Kris Ven, and Jan Verelst

Department of Management Information Systems, University of Antwerp,
Prinsstraat 13, B-2000 Antwerp, Belgium
{philip.huysmans,kris.ven,jan.verelst}@ua.ac.be

Abstract. Contemporary organizations are operating in increasingly volatile environments and must be able to respond quickly to changes in their environment. Enterprise Engineering aims to design agile organizations by applying theories from other fields. A theory from system sciences that receives much attention in this regard is modularity. In this paper, we apply a traditional modularity approach to a real-life case study. The subject of this case study is an enterprise architecture project in an e-government context. Our analysis shows that modularity can provide a relevant perspective in such projects. Moreover, it provides insights concerning the usage of enterprise architecture frameworks. However, due to the different nature of organizations, additional research is required to precisely understand the application of organizational modularity.

Keywords: Enterprise Architecture, Modularity, Enterprise Engineering.

1 Introduction

Contemporary organizations are operating in increasingly volatile environments and must be able to respond quickly to changes in their environment in order to gain a competitive advantage [1,2]. Moreover, these environments become increasingly complex. Consequently, it is hard to achieve long-term competitive advantages. Many authors argue that organizations therefore need to implement innovations at a steady pace to seek business sustainability [3], instead of relying on a single competitive advantage. The strategy organizations use to thrive in their environment is implemented in various organizational artifacts, such as data, processes, departments, and decision structures. Consequently, these artifacts need to be able to be adaptable when a strategy changes in response to changes in the environment. However, these artifacts, as well as their integration, increase in complexity, which makes them harder to adapt. As a result, organizational architectures are often perceived to be rigid instead of flexible.

Recently, modularity theory has been used by many authors to provide a scientific foundation for research concerning agile organizations. However, attempts to formulate a general theoretical framework for modular organizations often lead to conflicting results. In our research, we therefore use a bottom-up

A. Albani, J.L.G. Dietz, and J. Verelst (Eds.): EEWC 2011, LNBIP 79, pp. 106–120, 2011.

perspective, and study the application and relevance of modularity theory in real-life projects. In this paper, we present an exploratory case study in which a project is observed to introduce flexibility in governmental processes. While the project is considered to be successful, it remains difficult to objectivate the amount of flexibility which has been reached, or if improvements can be made. By analyzing the project using a modularity perspective, we attempt to objectivate the implemented solution. Based on the lessons learned during this research project, a better understanding of a relevant application of modularity in enterprise architecture projects can be achieved. We believe that such lessons can contribute to the emerging field of Enterprise Engineering [4].

The rest of this paper is structured as follows. In Section 2, we introduce the modularity literature which is used in the analysis. A description of the organization which is the subject of this case study is provided in Section 3. The enterprise architecture project is then analyzed from a modularity perspective in Section 4. Next, the implications and lessons learned are discussed in Section 5. Finally, conclusions and implications from this case study are discussed in Section 6.

2 Theoretical Foundation

Modularity is a concept from systems theory that has already been applied successfully in several domains. A common theme underlying the concept of modularity in each of these domains is achieving agility and flexibility [5,6]. The idea behind modularity is that a system should be composed in such a manner that all components are loosely coupled. To this end, system elements that must intensively interact with each other should be isolated in a separate module to ensure that changes to this module do not have an influence on other modules in the system. Communication between these modules is managed by well-defined interfaces [6]. The concept of modularity has been used in, amongst others, software engineering [7] and product design [8,9]. Campagnolo and Camuffo [10] provide a management literature overview which shows increasing attention to modularity in the organizational context. They confirmed research interest in organizational modularity by identifying a large number of publications concerning not only product modularity, but also modularity in production systems and the organizational structure itself. In this section, we first outline relevant literature concerning organizational modularity. Second, we discuss enterprise architecture frameworks, which are frequently used as the tool to design modular organizations. Third, we introduce the modularity approach we use in this paper.

2.1 Organizational Modularity

We focus on the use of modularity in the design of organizations [5,10,11]. This research area investigates how organizations can be constructed using loosely coupled and autonomous organizational artifacts that allow organizations to

adapt more quickly to changing environments [5,12]. This approach is compliant with the Enterprise Engineering research paradigm, which applies relevant theories from other fields to the evolvability of organizations. Much debate is going on concerning the exact implications of modularity. Many publications discuss the impact of the specificity of the market or product on modular organizational structures [13]. While certain authors assume a natural convergence of industries towards modular configurations [14], others claim that no clear evidence exists that such convergence is taking place [15]. Depending on the observed factors and the interactions which are taken into account, a top-down approach seems to lead to conflicting results. These conflicting results show that modularity has to be approached as a multifaceted concept [13]. We agree that organizational modularity has to be studied taking into account the context and specificity of the organization, as argued by [16]. This view suggests a bottom-up research approach, as opposed to a top-down analysis of markets and sectors. Therefore, it seems logical to study modularity in a real-life context, where influences of various factors can be analyzed, in order to deepen the understanding of modular organizations.

2.2 Enterprise Architecture

Enterprise architecture frameworks are proposed as a tool to design flexible organizations. They present an integrated view of the strategic goals and organizational and technical artifacts of an organization. By offering separate views for, e.g., process and data stakeholders, they enable separate analysis on these levels. Consequently, they seem to enable to view the organization as a set of building blocks which can be adapted independently. Enterprise architecture frameworks are therefore regarded as a tool to implement and maintain evolvable organizations. However, a clear distinction should be made between a structure of loosely coupled artifacts and a collection of models with specific abstractions. Consider a tightly integrated information system. It is possible to create different models of such a system which only represent certain aspects and abstract others. Such models can be useful for, e.g., improved understanding of different parts of the organization, or to create a common language for communication. However, when changes to this system need to be made, it is no longer possible to neglect all abstracted aspects, since the modeled aspects might heavily impact aspects which are invisible in the model. In contrast, a system with loosely coupled modules may be more complex to represent, but will be easier to adapt, since the impact of a change will be contained in the module itself. While many authors cite modularity as the underlying theory for constructing enterprise architectures, few reports exist which analyze real-life solutions from an explicit modularity perspective. Some enterprise architecture frameworks indeed seem to focus solely on representing models, but neglect the actual structure of the artifact. In our opinion, the actual artifact implementation cannot be neglected in an Enterprise Engineering viewpoint.

2.3 Baldwin and Clark

Given the common theoretical foundation to study modular artifacts, it has been proposed to analyze modular organizations with similar tools as those used for analyzing other modular artifacts. Baldwin and Clark [6,17,18] propose to design modular artifacts using a technique called Design Structure Matrix (DSM) Mapping. In a DSM, an artifact is described by a set of design parameters. The matrix is then filled by checking for each parameter by which other parameters it is affected and which parameters are affected by it. The result is a map of dependencies that affect the detailed structure of the artifact. Dependencies are represented by an "x". The intersection of identical design options is marked with a ".". An example design structure matrix is shown in Table 1. Consider the design dependency which is represented by the "x" in the intersection of the column of design option A2 and the row of design option A1. This signifies that design option A2 influences design option A1: the design decision for design option A1 will be dependent on the decision taken for design option A2. This dependency does not break the modular structure of the artifact, since design options A1 and A2 both belong to the same module. Now consider the dependency of design option B1 on design option A2. Since these design options belong to different modules, it can be concluded that these modules are directly dependent on each other. Therefore, this dependency violates the modular structure. Indirect or chained dependencies can occur as well. While design option B2 does not seem to affect any design options of module A, it does affect design option B1, which in turn affects design option A1. Therefore, a so-called chained dependency exists between design options B2 and A2.

Table 1. Example design Structure Matrix

		Module A		Module B	
		Option A1	Option A2	Option B1	Option B2
Module A	Option A1	.	x		
	Option A2		.	x	
Module B	Option B1			.	x
	Option B2				.

Baldwin and Clark state that "[b]ecause of these dependencies, there will be consequences and ramifications of any choice" made during the design of the artifact [6]. A design choice for a given parameter can limit or affect the possible design choices concerning other parameters. Moreover, when the artifact needs to be changed, interactions between design options can alter the performance of the artifact, or even break its functionality. Therefore, if a certain design parameter is changed, the impact of this change on all design parameters it affects needs to be accounted for. Consequently, changes to an artifact with an

imperfect modular structure cause ripple effects throughout the design. As a result, the magnitude of the effort needed to apply a certain change becomes unpredictable.

According to Baldwin and Clark [6], dependencies in a DSM can be resolved through the process of design rationalization. Instead of allowing different design options for a given parameter, a *design rule* could be defined which enforces the use of a certain design option. Doing so restricts the design alternatives, but prevents uncontrolled ripple effects between the various modules. This is in line with the definition of architecture used in Enterprise Ontology, which states that architecture limits design freedom [19]. The use of such design rules can be observed in the various application fields of modularity. A *standard* in the computer industry could be considered as a bundle of design rules. For example, the TCP/IP standard specifies communication protocols for computer networks, and therefore limits the choice of protocols on each network layer. In an organizational context, the use of such design rules is proposed by *prescriptive* enterprise architectures. The design rationalization process results in a structure of global design rules, which have to be adhered to by all modules, and a set of modules which should be independent from each other.

3 Case Description

In this case study, we focus on a Belgian governmental organization, whose mission is to introduce and implement e-government solutions. To achieve this goal, it undertakes projects in the field of back-office reengineering, and tries to leverage these improvements by supporting projects with governmental partners. In this paper, we analyze a project that improves the way data from various sources is used in governmental processes. We will refer to this project as the Data Usage in Governmental Processes (DUGP) project.

The need for the DUGP project arises from the complex collection of data sources used in governmental processes. Because of the different governments on the federal, regional and community level, different data sources fall under the responsibility of different governmental entities. As a result, the data in these data sources is offered in different ways. The data sources which need to adhere to laws and decrees are referred to as authentic data sources. Governmental organizations that offer their services through processes (which we will call process owners) may rely on data from different sources. For example, the process to request construction premiums requires personal data of the citizen requesting the premium (from data source A) as well as geographical data of the construction site (from data source B). Since the databases which contain the personal and geographical data are not integrated or standardized, various conversions between implementation aspects of these data sources may be necessary. Suppose that the information required from data source A needs to be obtained by invoking a single web service call, providing the address from the end user using four data fields (street name, street number, bus number, city name). In order to query geographical information in data source B, a request file containing two data

fields (street name and street number, and ZIP code) has to be transferred using the FTP protocol. Since the construction premium process depends on both data sources, it needs to be able to communicate using two different versions of address data syntax. The complexity of these conversions is a barrier for the use of these data sources. As a result, many processes are designed to request the required information from the end user. Different process owners therefore create custom databases which contain information that is available in authentic data sources as well. However, the quality of such data sources cannot be checked or guaranteed. For example, a citizen who changes his address is only required to notify official instances of this change. Consequently, an organization using a custom database to save addresses instead of requesting them from the official data source will eventually have incorrect data.

The goal of the DUGP project is twofold. First, process owners need a more uniform and simpler way to access authentic data sources. Currently, the difference in accessing data from various providers leads to process owners who collect the required data themselves, instead of requesting it from authentic data sources. A central platform which offers a uniform way to access data is expected to remedy this. Moreover, a central platform enables more transparent data management: collection and administration of data is a concern of the responsible entity; technical, legal and organizational aspects which are common for all data sources would be addressed by organization responsible for the central platform. Second, the DUGP project aims to improve the integration of data between data sources. This would improve the understanding of relationships between various data sources, which would be beneficial for process designers. Currently, such integration is missing since the data sources belong to organizations in different levels of the governmental hierarchy, and no organization is responsible for the explicit integration of data. In order to promote adoption of the platform, the e-government organization will co-finance projects which use the platform to provide better integration and more efficient processes. Given the broad scope and the required interaction with other governmental organizations, an enterprise architecture framework was selected to guide the DUGP project. The organization uses an adapted version of the Zachman framework to define the needs for their projects and guide the development and implementation. In their framework, a column was added for governance, and rows were added for testing and deployment of artifacts. Based on this framework, the organization decided to build a platform to offer unified and integrated access to authentic data sources. The platform is based on two existing data sources from the federal government. Data from these data sources will be augmented with data available in data sources from other governments (e.g., geographical data, which is offered by regional governments). The first data source which is used focuses on data concerning organizations. In this data source, data such as registration number, official addresses and legal statute of enterprises can be obtained. We will refer to this data source as the Data Source for Organizations (DSO)[1]. The Federal Public Service Economy is responsible for this data source. The second

[1] In order to preserve anonimity, DSO, EDSO, DSI and EDSI are fictional names.

data source offers data concerning individuals. It refers to data such as employment and social status of citizens. We will refer to this data source as the Data Source for Individuals (DSI)[1]. This data source is governed by a separate organization created by the federal government. The platform will maintain this distinction, and offer its data services grouped in an Enhanced Data Source for Organizations (EDSO)[1] and an Enhanced Data Source for Individuals (EDSI)[1].

Since the EDSO relies on the DSO for its data, and the EDSI relies on the DSI, they need to consider the implementation of these data sources. Many implementation aspects are quite different. For example, the DSI has webservices available to query its data. As a result, these webservices can be used to develop webservices in the EDSI. These webservices are not directly offered in their original form. Instead, a facade pattern is used. This enables the creation of a uniform web service syntax throughout the platform. In contrast, the DSO has no webservices available. It is a mainframe which operates using batch requests. Therefore, a copy is made from the original DSO every night. This copy is then augmented with data from other governmental authorities, and used as a central database on which the services from the EDSO are provided. In order to simplify data access, the new platform provides three data delivery methods which will be available for all data sources: data repositories, an online application and webservices. Customized data repositories are large data files, which are copied to the process owner. After this initial data provision, automatic updates are transferred when data changes. These repositories are offered to enable process owners to incorporate the data from the new platform in their processes, without having to implement a webservices-based data access. Since many organizations are accustomed to using their own data sources in their processes, a customized data repository can be implemented without many changes in the processes. However, the unauthorized data sources which have been collected by the organizations themselves will then be replaced with authentic data. The online application allows for manual consultation of the data with a much smaller granularity: instead of a single large data file, only the result of a single query is returned. The same result can be obtained automatically through the use of webservices. Webservices offer the same data granularity, but can be implemented to automate processes.

4 Application of Modularity

In this section, we analyze the DUGP project presented in Section 3 using the theory outlined in Section 2. First, we attempt to gain a relevant understanding of the identified issues using a modularity perspective. By expressing the problem description using terminology from modularity literature, we attempt to clearly specify which issues will be resolved in the project. Second, we assess whether the guidelines suggested in the modularity literature can be applied in this case study, and compare them with the solution which was proposed in practice. Third, we evaluate the built artifact using modularity criteria. We are then able to motivate this design in a more structured way, and possibly identify improvements to the design.

4.1 Modeling Data Sources and Processes before the DUGP-Project

Before the introduction of the platform, usage of the different data sources by process owners was not straightforward. Usage of different data sources resulted in multiple implementations of the same functionality. Consider the implementations for data syntax and data delivery from the construction premium example introduced in Section 3.

Table 2. Design Structure Matrix before the project

		Process				DS			
		Throughput	Data Retreival	Data Syntax	Data Dictionary	Capacity	Data Delivery	Data Syntax	Data Dictionary
Process	Process Throughput	.	x			x			
	Data Retrieval		.	x		x	x		
	Data Syntax			.	x			x	x
	Data Dictionary			x	.			x	x
DS	Capacity					.	x		
	Data Delivery					x	.	x	
	Data Syntax						x	.	x
	Data Dictionary							x	.

If we interpret this example according to modularity theory, these issues manifest as design dependencies in a design structure matrix. The dependencies show which implementation aspects of a data source influence the process design, and which aspects only impact the internal design of the data source. Consider the partly DSM for the construction premium example process in Table 2. In this table, processes and data sources (DS) are considered to be modules. A sample is taken from the aspects of data and process implementations which have been mentioned in the case study interview. These aspects are represented as design parameters. The "x"-es on a grey background represent the design dependencies discussed in Section 3. The data syntax used in the process depends on the data syntax used by the data source, and data retrieval of the process depends on the data delivery of the data source. Since every single process needs to access data directly from different sources, the data syntax used in that process is dependent of every data source it uses. Therefore, design dependencies can exist between the process modules and multiple data source module instances. When data syntax differs between two data sources, the *process* has to be capable of working with these two different syntaxes, for example by providing a conversion mechanism. The same reasoning goes for data delivery: the *process* has to be able to interpret XML responses from webservices, as well as read physical files delivered by FTP. In other words, the process needs to be aware of the possible implementations of different aspects of the data sources, and account for all

these variations. Implementing and maintaining multiple versions for all aspects results in additional complexity, which is not inherent to the process itself.

This DSM shows that various aspects need to be considered for both data sources. However, when additional data sources are needed, additional dependencies will be added. Consequently, a process change which requires an additional data source can result in additional implementations of all the mentioned data design parameters on which aspects of the process module depends. Moreover, additional implementations may be necessary when a data source changes a certain design option. The occurrence of design dependencies therefore can force a module to adapt to changes in modules outside its control.

Using the design structure matrix, we can explicitly identify sources of interactions between modules, and explain why complexity exists in the integration of these modules in systems. Doing so objectivates the issues which need to be addressed by the DUGP project. Moreover, while the same issues have been identified by knowledgeable and experienced employees, an analysis using DSM seems to be more structured, and therefore more repeatable. A respondent confirmed that "while these dependencies are known as critical problems in the heads of our experienced analysts, we have no way of documenting them, or capturing them when an analyst leaves our team."

4.2 Solutions Proposed by Modularity

The illustrated dependencies reveal that relevant modules can be identified, and that it makes sense to consider the data and processes as a modular structure in order to provide a precise problem description. The modular structure of processes and data sources before the DUGP project was tightly coupled, as shown by the identified dependencies. The goal of the project is thus to enable the modules to be more loosely coupled. Only then, traditional modularity benefits and modular operators can be applied. For example, modularity promises to enable concurrent upgrade processes. In this case, processes will be able to implement changes without being restricted by implementation aspects of data sources. The specification of a new process to realize e-government goals (adding a module to the modular structure), or a reorganization of the data sources because of new regulations (splitting modules) will not interfere with the operation of existing processes.

According to Baldwin and Clark [6], architectural rules have to be defined in order to eliminate dependencies. By restricting design freedom, a uniform solution can be enforced. In this case, this would eliminate the need to use multiple implementations for a given design option. For example, data delivery could be restricted to web services, or a reference model could be imposed which ensures a common data syntax. For every dependency which occurs between two different modules, an architectural rule needs to be defined. If all dependencies are handled this way, design dependencies will only occur within modules. Architectural rules need to be specified by a central organizational unit which has authority on the level and scope which the rule applies to. Here, the responsibility for implementing the design options according to the architectural rules can

be attributed to the organizations responsible for the data sources. Because of the political situation, the various data sources fall under the responsibility of different governmental entities. Currently, no central organizational entity exists which could decide and impose architectural rules to the organizations governing the data sources. A respondent agrees that "central control is necessary to impose such rules". He further notes that "however, the government resembles more a constellation of SME's, instead of one large organization." An agreement between all governmental units would need be reached to establish any architectural rule. However, most organizations have legacy systems, which are difficult to change. As a result, they are not inclined to accept the introduction of such architectural rules. Consequently, we can conclude that the traditional solution suggested by modularity theory is not feasible in this case.

In order to be able to offer data access which adheres to architectural rules, an intermediary platform needs to be built, which will be controlled by the e-government organization. The platform which is build can therefore be considered to be an additional module, which adheres to architectural rules defined by the organization. In the following subsection, we analyze this platform from a modularity viewpoint.

4.3 DUGP Project Analysis

To analyze the implemented solution as a modular structure, we consider processes, data sources (DS) and the developed platform as modules. These modules are shown in the design structure matrix in Table 3. The goal of the DUGP project is to eliminate dependencies between the data source and process modules. When comparing the platform with the design structure matrix in Table 2, we can conclude that some of these dependencies are indeed eliminated. Consider for example the data syntax and data delivery. We included an empty grey background to mark the previous existence of these dependencies. The syntax of webservices offered by EDSI is decoupled from the naming conventions of the DSI by using a facade pattern. As a result, naming conventions can be kept internally consistent with custom-built webservices for EDSO. The data syntax can be considered as an architectural rule which is maintained by the e-government organization. By adhering to this data syntax, process owners no longer need conversions between different data syntax versions. Another example is the data delivery design. In data sources from the platform, data can be provided through customized data repositories, an online application or webservices. Here, a single design option has not been selected, but the consistent offering of all data delivery techniques allows process owners to implement a single design for data delivery. Again, process owners no longer depend on the specific data delivery technique of the individual data sources. Consequently, it seems that the platform aids to decouple the process owners from the design decision of the data sources.

However, as stated by Baldwin and Clark, eliminating all dependencies in a modular structure is not a trivial task [6]. Consider the design option *process throughput* in the case of an automated process. Our respondents indicated that

Table 3. Design Structure Matrix for the developed platform

		Process				Platform				DS			
		P. Throughput	Data Retrieval	Data Syntax	Data Dictionary	Capacity	Data Delivery	Data Syntax	Data Dictionary	Capacity	Data Delivery	Data Syntax	Data Dictionary
Process	Process Throughput	.	x			**x**							
	Data Retrieval	x	.	x		x							
	Data Syntax	x		.	x				x				
	Data Dictionary				.				x				x
Platform	Capacity					.	x			x	x		
	Data Delivery					x	.	x		x			
	Data Syntax						x	.	x			x	
	Data Dictionary				x			x	.			x	x
DS	Capacity									.	x	x	
	Data Delivery									x	.	x	
	Data Syntax										x	.	x
	Data Dictionary											x	.

the number of processes which can be supported is limited by, amongst others, the capacity of the data delivery implementation. In Table 3, this is visualized by the "x" where the column of the capacity of the platform intersects with the row of process throughput. A possible data delivery implementation in the platform are webservices. As described above, webservices from the platform can be either custom-built by the e-government organization (e.g., webservices for EDSO), or can be part of a facade-pattern, calling underlying webservices (e.g., webservices for EDSI). The custom-built webservices operate on a local database. Consequently, their capacity is limited by the servers of the e-government organization itself. However, webservices which are part of the facade pattern are dependent on the capacity of the underlying services of DSI. Based on the implementation, issues with webservice capacity need to be discussed with the e-government organization or with the organization responsible for the original data source. The difference between the implementation of webservices in the platform is based on the available data delivery techniques from the original data sources. In Table 3, this is visualized by the "x" where the column of the data delivery of a data source intersects with row of the capacity of the platform. It therefore seems that an unexpected dependency can be identified: when the platform is used, the process throughput design decision is impacted by the data delivery technique of the original date source. When the data delivery of DSO is changed (e.g., webservices become available) and used by the platform, process performance may be impacted. This is an example of a chained design dependency which propagates through the design structure matrix. Such dependencies are difficult to trace and to account for in change projects.

5 Discussion

The project under review aims to decouple organizational artifacts in order to introduce flexibility for governmental organizations. By introducing a platform as an architectural layer between data sources and processes, the impact of a change on the process level or variability on the data level is kept minimal. Such projects are usually perceived to be rather complex. In this case study, we explored the use of an approach using traditional modularity tools. We illustrated our approach with partly design structure matrices to illustrate the key contributions of this approach. While we do not claim to be able to generalize our results based on a single case study, we conclude that a modularity approach proved useful in this project. More specifically, we identified following advantages.

First, the identified dependencies in the design structure matrix allow for a more structured and objective analysis of the problem description. In Section 4.1, it is shown that the occurrence of dependencies between the process and data source modules introduce complexity from the data sources to the process level. Since this complexity does not belong to the process level, it is perceived by process owners as an important barrier for the use of the data sources. Therefore, the data sources were underused, and process owners relied instead on the creation and use of unauthorized data sources. Put differently, these dependencies show that the data sources could not be used as black box modules. The design options which have an impact on the process module indicate which aspects necessitate a white box view on the data sources. Enterprise Engineering advocates a strict distinction between these two views [4]. However, the occurrence of design dependencies prohibits the use of a strict black box view. Second, additional dependencies have been identified through the structured analysis of the design structure matrix. By identifying relevant design options and checking their interaction with design options of related modules, issues can be discovered which are not clearly present. Indeed, without an explicit modularity perspective, it is unlikely that a perfect modular structure would be designed. More specifically, the occurrence of chained dependencies can be expected to be discovered more easily through an analysis using a design structure matrix. Third, expected risks when implementing a certain design can be estimated more easily. For example, modularity theory suggests to eliminate existing dependencies by specifying architectural rules. By selecting a single option for a design option, all modules can assume a stable design decision for that option. However, for the data delivery design option, the choice was made to support three different design options (custom repositories, an online application and webservices). This design decision deviates explicitly from the solution proposed by modularity. Consequently, additional effort will need to be attributed to this design option to prevent it from impacting the process module. Concretely, the e-government organization needs to keep supporting three different data delivery implementations equally, instead of a single one.

Nonetheless, the analysis shows that the DUGP project indeed improved modularity. Moreover, the identified benefits indicate the this modularity approach can be useful in such projects. However, the nature of organizations is quite

different from other artifacts which modularity has been applied to. Therefore, it can be expected that an identical application of modularity as in other fields is not feasible. In this case study, it is clear that the division of authority among various organizations made the specification of organization-wide architectural rules impossible. Consequently, the e-government organization needed to introduce an additional module to eliminate the direct dependencies between data sources. The developed platform therefore needs to interact with data sources, and present the required functionality to process owners without exporting non-relevant aspects. Many dependencies have been eliminated by this solution. However, the disadvantage is that the creation of a new module which interacts with other modules can also introduce new dependencies. Because of the integration with multiple modules, especially the risk for introducing chained dependencies increases. These dependencies are much harder to identify and to remedy than direct dependencies. Moreover, the introduction of the e-government organization as an intermediary organization makes the structure more complex and less transparent: we discussed how in case of poor process throughput, it is unclear which organization needs to be addressed.

The lessons learned from this case study can be used to understand how enterprise architecture frameworks can support complex projects. Consider for example the organizational processes in this case study. While they can be separately represented in a process model view in an enterprise architecture framework, their concrete implementation incorporates many aspects of elements from other organizational levels. While analysis and communication can be supported by process models, it is, ultimately, this implementation that needs to be adapted. Exactly these interactions are made explicit using a modularity approach. By focussing on decoupling modules, the complexity which is not inherent to that level is eliminated. If all dependencies with modules from other levels could be eliminated, only the complexity inherent to the process level would remain. However, the general use of modularity tools such as the design structure matrix may be too vague to apply directly in enterprise architecture projects. Therefore, guidelines are needed to support the design of decoupled organizational artifacts. In an organizational context, the use of such design rules is proposed by *prescriptive* enterprise architectures. In a prescriptive enterprise architecture, principles have to be defined by the organization, which have to be adhered to during the design of any organizational artifact. This is in line with the definition of architecture in Enterprise Engineering, which states that "an architecture consists of normative principles which guide the development of an enterprise" [4]. In most frameworks, these principles are considered to ensure the implementation of an organization-specific strategy. Consequently, it is assumed that such guidelines need to be created by the organization itself. Currently, no design rules or principles have been proposed to achieve certain characteristics independent of the organization. For example, no guidelines exist in prescriptive enterprise architecture frameworks to design an evolvable organization. Such guidelines are not present in *descriptive* enterprise architecture frameworks either. When positioning the DUGP project in an descriptive enterprise architecture framework, it

seems to have a clear focus. Consider the Zachman Framework. The complexity addressed in this project is located in the *logical* and *physical* layer of the *data* column. In the *logical* layer, the specific data entities and data relationships are described. The implementation of the entities and relationships is described in the *physical* layer. The different methods to deliver the data would be described in the *function* column of these layers. Positioning these elements in the framework could help to identify relevant module candidates, but does not provide any insight regarding design dependencies. The Zachman framework only claims to be a taxonomy to position organizational models, and does not define any requirements for the specific models. The ability to identify design dependencies has to be provided by a relevant modeling method. Therefore, we can conclude that the approach used in this paper is supplementary to the use of prescriptive and descriptive enterprise architecture frameworks.

6 Conclusion and Future Research

This paper illustrates the potential of Enterprise Engineering: knowledge which exists and is applied in other fields can aid to remedy organizational issues. The current methods which are used in this context are usually not supported by an integrating theory, which inherently limits their general applicability. In our research, we consider modularity as a candidate for such an integrating theory. However, organizations are, by nature, quite different than artifacts traditionally used in modularity approaches. Therefore, researchers should focus more on the precise implications of modularity. For example, this case study illustrated that an organizational structure cannot be considered to be modular just because it consists of separate organizations. If no architectural rules are applied to ensure consistent interoperability, none of the benefits associated with modular structures will be achieved. In [10], it is indeed argued that "controversies and ambiguities on how modularity should be defined, measured and used in managerially meaningful ways" still exist. We will therefore continue to use a bottom-up perspective in our future research. By analyzing additional projects on different organizational levels, we strive for a better insight of organizational modularity and its impact on the use of enterprise architecture frameworks.

References

1. Teece, D.J., Pisano, G., Shuen, A.: Dynamic capabilities and strategic management. Strategic Management Journal 18(7), 509–533 (1997)
2. Eisenhardt, K.M., Martin, J.A.: Dynamic capabilities: What are they? Strategic Management Journal 21(10/11), 1105–1121 (2000)
3. Van de Ven Andrew, H., Harold, L.A.: An Introduction to the Minnesota Innovation Research Program. Oxford University Press, New York (2000)
4. Dietz, J.L.G.: Enterprise Engineering Manifesto., http://ciaonetwork.org/publications/EEManifesto.pdf
5. Sanchez, R., Mahoney, J.T.: Modularity, flexibility, and knowledge management in product and organization design. Strategic Management Journal 17, 63–76 (1996)

6. Baldwin, C.Y., Clark, K.B.: Design Rules. The Power of Modularity, volume 1 of MIT Press Books, vol. 1. The MIT Press, Cambridge (January 2000)
7. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM 15(12), 1053–1058 (1972)
8. Salvador, F., Forza, C., Rungtusanatham, M.: Modularity, product variety, production volume, and component sourcing: theorizing beyond generic prescriptions. Journal of Operations Management 20(5), 549–575 (2002)
9. Fixson, S.K.: Product architecture assessment: a tool to link product, process, and supply chain design decisions. Journal of Operations Management. 23(3-4), 345–369 (2005); Coordinating Product Design, Process Design and Supply Chain Design Decisions
10. Campagnolo, D., Camuffo, A.: The concept of modularity in management studies: A literature review. International Journal of Management Reviews (2010)
11. Tiwana, A.: Does interfirm modularity complement ignorance? a field study of software outsourcing alliances. Strategic Management Journal 29(11), 1241–1252 (2008)
12. Douglas Orton, J., Weick, K.E.: Loosely coupled systems: A reconceptualization. The Academy of Management Review 15(2), 203–223 (1990)
13. Hoetker, G.: Do modular products lead to modular organizations? Strategic Management Journal 27(6), 501–518 (2006)
14. Galvin, P., Morkel, A.: Modularity on industry structure: The case of the world the effect of product bicycle industry. Industry & Innovation 8(1), 31–47 (2001)
15. Chesbrough, H.W., Kusunoki, K.: The modularity trap: innovation, technology phase-shifts and the resulting limits of virtual organisations, pp. 202–230. Sage, Thousand Oaks (2001)
16. Berger, S.: How We Compete: What Companies Around the World Are Doing to Make it in Today's Global Economy, doubleday (2005)
17. Baldwin, C.Y., Clark, K.B.: The value, costs and organizational consequences of modularity. Working Paper (May 2003)
18. Baldwin, C.Y.: Where do transactions come from? modularity, transactions, and the boundaries of firms. Industrial and Corporate Change 17(1), 155–195 (2008)
19. Dietz, J.L.G.: Enterprise Ontology: Theory and Methodology. Springer, Heidelberg (May 2006)

Author Index