



From Commitments and Responsibilities to Evolvable Software

Wim Ploegaerts

1	The Methodologies	5
1.1	Some terminology	6
1.2	Design and Engineering Methodology for Organizations (DEMO)	8
1.3	MERODE	11
1.4	Normalized Systems (Theory)	15
1.5	Things to read	19
2	Case Study IES	23
2.1	Introduction	24
2.2	Case Description	26
2.3	Discussion of the EIS Solution	35
2.4	Conclusion	52
	Bibliography	53
	Books	53
	Articles	53

Introduction

The idea to write this document was born when reviewing the different chapters of Jan De Winter's book "Concise Guide To Enterprise Transformation". I saw it gradually grow from the initial idea formulated when Jan was still quarantined in the hospital to the hard copy with his name on it. I noticed that writing a book requires a lot of patience and is really hard work. One of the most difficult problems to solve seems to me the scope definition. What should be in the book because it strengthens the message versus what you think is very interesting and important but may even confuse the reader.

This document can be considered as a chapter that did not make it in the final selection of the content of the book. It starts where the book stops. The book explains how you can reach a shared mental model underpinning an Enterprise Transformation and how you can translate this in a precise model documenting the responsibilities and flow of commitments required to generate the intended output of the transformed organization. The next step, not described in the book, is the translation of this model into systems and real executable processes.

I do not pretend that the approach described in this document is the only or even the best approach. The approach presented is however feasible. It can be an evolvable and maintainable software system faster than when using a classical approach. The approach is based on a collection of methodologies and theories with a sound theoretical basis. At the same time, all these methodologies have already proven that they can be used in real life complex project and not only for toy applications.

The combination of the methodologies presented focuses on the development of software systems ("coding"). It should be realized that only in a limited number of greenfield projects one has the luxury to implement the entire required scope from scratch. Most often, the organization already has a complex system landscape supporting the existing operations. Therefore, in many cases, the next step is to map the new requirements onto the existing landscape, determine what can (or must) be reused, what needs to be changed and which parts require new developments. In many cases, a successful approach that is overlooked is a fundamental redesign of the interface between existing systems or, by extension, to build an interface layer that becomes the owner and "orchestrator" of (part of) the new requirements. Enterprise Engineering can help with this too. How to do this may become part of another document.

One of the main advantages of the proposed approach is that it is entirely wrapped around an essential success factor for IT projects: **communication**. True business-IT alignment can be achieved by using the proposed modeling approach. The resulting models can be transformed into working code using code expanders. This drastically lowers the implementation time and results in short development cycles, supporting a truly agile approach allowing the business people to review the impact of their decisions.

The code expanders we propose to use are based on the Normalized Systems Theory and deliver truly evolvable software systems. This guarantees that an incremental build of the system will not impact the quality of the code base. It also guarantees that the future introduction of new technologies and platforms can be done at a reasonable cost.

The four theories and methodologies used are the following:

- Appreciative Inquiry
- Enterprise Engineering and its *Design and Engineering Methodology for Organizations*
- The MERODE methodology and its software architecture
- Normalized Systems Theory and its expanders

In the first chapter of this document, these methodologies will be discussed briefly. It is by no means the goal to discuss the methodologies in depth. The interested reader is referred to the books, academic articles and course material that is available.

In a second chapter, the *Import Export Services* case will be discussed. This is a more advanced case used in DEMO trainings. The concepts mentioned in the first chapter will be applied and explained in a more practical way and additional concepts of the different methodologies will be introduced whenever required.

It should be noted that Appreciative Inquiry and its links with Enterprise Engineering is extensively handled in the book of Jan De Winter [1]. I do not have the ambition nor the knowledge to make an addition to it. As already mention above, **in this document we pick up the story where the book stopped.** This is the point were all parties involved have agreed on the organization's "cooperation model of the future" and decided to go forward with the implementation.

1. The Methodologies

1.1 Some terminology

1.1.1 Model driven engineering/architecture

As can be found on Wikipedia *Model-driven engineering (MDE) is a software development methodology that focuses on creating and exploiting domain models, which are conceptual models of all the topics related to a specific problem. Hence, it highlights and aims at abstract representations of the knowledge and activities that govern a particular application domain, rather than the computing (i.e., algorithmic) concepts.*

MDE should not be confused with MDA Model Driven Architecture. The latter is a specific approach defined by the Object Management Group (OMG) beginning of the 2000s. See also www.omg.org/mda for more information on MDA. The goal is to separate business logic from implementation details. The required business logic is described in a "Platform Independent Model" (PIM). The PIM is transformed in to a "Platform Specific Model" (PSM) for a specific platform or technology. This transformation can be fully automatic or semi-automatic.

MDE is a broader paradigm as it uses models at various stages of the software engineering lifecycle; not just at the architectural level, but also at the level of requirements, design, implementation, testing, and maintenance. The goal is to improve the productivity, quality and maintainability (evolvability) of the software development process. As there are multiple model types, there can be different types of transformation, e.g. from one model to another or from a model into code.

1.1.2 Domain modelling

Domain modeling is the process of representing concepts, entities, relationships, and the rules that are relevant to a specific problem. The goal is to capture the essence of a particular area of expertise (e.g., banking, insurance, e-commerce), completely independent of any technical implementation. The outcome of this exercise is a *domain model*, which is often described in a diagram using some graphical notation like UML model or a DEMO model. The domain model is used to create or to reflect the shared understanding among stakeholders. It is the starting point for further (detailed) analysis, design and later the implementation. The domain model is an essential part of Model Driven Engineering. In MDA, the domain model is typically part of the PIM.

1.1.3 Domain Specific Language (DSL)

A **Domain-Specific Language (DSL)** is a notation or language, designed to solve problems in a particular domain and tailored to the specific needs of that domain. A DSL can also be a programming language. Unlike general purpose programming languages like C++ or Java, DSLs have been designed to optimally express the concepts used in a specific domain. This makes the program easier to write and to understand.

An old DSL used in the domain of the design of integrated circuits is *VHDL*. It dates from the eighties. According to Wikipedia, VHDL is a hardware description language that can model the behavior and structure of digital systems at multiple levels of abstraction, ranging from the system level down to that of logic gates. The main advantage of VHDL, when used for systems design, is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). Another benefit is that VHDL allows the description of a concurrent system. Only designers of integrated circuits would ever consider to use VHDL. No general purpose programming language can replace the power of VHDL. The building blocks of VHDL ("statements") all have a clear semantics that can be used to simulate the behavior of the circuit that is described (electrical, logical, behavioral)

DSLs are an essential part of MDE and MDA. They are essential for the transformation. Every construct in the DSL captures a concept in the domain and will be transformed in a predictable manner to the next level (e.g. code or another model).

1.1.4 Enterprise model

According to Wikipedia, **Enterprise modelling** is the abstract representation, description and definition of the structure, processes, information and resources of an identifiable business, government body, or other large organization. It deals with the process of understanding an organization and improving its performance through creation and analysis of enterprise models. It serves as a high-level abstraction to help stakeholders understand, analyze, design, and improve the business as a whole. Key Aspects of an Enterprise Model. In its broadest sense, the enterprise model includes different aspects:

- The **Information Model** which captures the data entities, relationships, and flows that are crucial to the business. This might include customers, products, orders, and other key business objects
- The **Business Processes** describing the workflows, activities, and tasks that make up the organization's operations. Examples include order processing, customer support, or supply chain management.
- The **Organizational Structure** representing the roles, responsibilities, reporting relationships, and organizational units (e.g., departments, teams).
- The **Rules and Policies** describing business rules, regulations, and constraints that govern how the enterprise operates.
- The **Stakeholders and Interactions**, identifying internal and external actors (e.g., customers, suppliers, partners) and how they interact with the enterprise.

Note that while a domain model focuses on a specific business area (e.g., finance, HR), the enterprise model covers the whole organization.

An example from the banking industry is the creation of an enterprise model of the mortgage lending process. This model would include the following:

- The sequence of steps from customer inquiry to loan approval.
- The roles involved (loan officers, risk assessors, IT systems).
- The data that flows between these steps (application forms, credit checks, approval documents).
- The rules governing each step (regulatory requirements, internal policies).

1.2 Design and Engineering Methodology for Organizations (DEMO)

DEMO is a methodology that is based on the principles of Enterprise Engineering, as described in the books of Dietz and Mulder [2][3][4]. As the name of the methodology makes clear, it is not a methodology to develop software systems. It focuses on the design of Organizations, and more specifically on the design of the commitments and responsibilities that are required to produce the intended output of the organization. Contrary to many other methodologies, it does not focus on how the output is created. This is actually what makes this methodology unique and what makes it the ultimately fit for organizational redesign, especially in combination with Appreciative Inquiry.

To come to a shared mental model about the "organization of the future", many people need to be involved in the discussion. It is clear that top-management and C-level need to be involved, and not only the domain experts. If you ever tried to discuss an IT model with business people and especially with top-management, you know that it is not a good idea. There is a high probability that you will not have a very good response if you get one at all. Abstract IT models, like information and process models are, most of the time, not part of the world top-management is interested in. What is worse for people that want to present these model to top-management is that, in many cases, they do not know how things are produced, at least not in sufficient detail. For that they have domain specialists working for them.

What every top manager *does* know very well and is interested in are *commitments and responsibilities*, exactly the two things that are modelled in DEMO. In many cases you do not create the model with the C-level but with middle management and domain experts. The outcome can however be discussed with the decision makers. When for example, you encounter conflicting views during model creation you can ask top management for a decision. It is my experience that you *can* have in depth discussions about the content of the DEMO cooperation model with the decision makers.

1.2.1 Commitments and responsibilities

The core of DEMO and enterprise engineering is the business transaction, which is a *commitment oriented communication between two responsible parties*. This is treated extensively in all books on enterprise engineering. To understand the full depth of the concept and its implications, I would advise the reader to start with [3]. The subject is also discussed in [1], but obviously from the angle of enterprise transformation.

The basic observation is that in an organization "things get done" because people commit to do things to realize the goals of the organization. It is observed that such a goal oriented communication always follows the same pattern. Let me use the example Jan De Winter and I like to use when explaining DEMO, and also used in [1]. Assume I go to a bakery to buy a large brown bread.

- First I **request** the baker a large brown bread
- Next he **promises** to deliver me the bread I asked for. This step can be implicit of course.
- Then he takes the bread - this is the **production** for which we have the conversation
- Next he gives me the bread and **states** "here it is"
- The final step in the transaction is that I take the bread, see that it is a large brown bread and I **accept** it, ending the transaction

This four step pattern is the "happy flow" of any commitment oriented communication. In nearly every step, the happy flow may fail and the parties may enter a *discussion*. For example, instead of promising to give me a large brown bread, the baker may respond by telling me he has none left, giving me the opportunity to order something else. It has been shown that also the discussions follow a fixed transaction pattern.

The business transaction in the example does not stand on its own of course. I assume that the

baker wants to be paid. Here it becomes interesting from an organizational design perspective. The organization can decide that the bread should be paid before we promise to deliver it (the webshop approach), or we can ask for payment after the bread was delivered. In certain cases you may want to support both approaches, e.g. the first approach for occasional customers, the second for trusted customers. If you take that approach, somebody will have to define what a trusted customer is.

1.2.2 The DEMO submodels

The DEMO model consists of four submodels

- Cooperation model (CM)
- Process model (PM)
- Fact model (FM)
- Action model (AM)

As discussed in depth in [1], the cooperation model is the submodel where you document the commitments and responsibilities. In every commitment you make, there is some information involved and/or produced. This is modelled in the fact model, which is an information model. For example in a webshop, if you commit to deliver an item the next day, you need to know at least which item needs to be shipped but of course whether the item will be available for shipment.

In nearly all cases the commitments are expected to occur in a specific sequence. Take the example of a coffee shop. Assume that I only commit to deliver you a coffee after you have paid. The intended relation between the commitments is modelled in the process model. The establishment of the PM is an important part of your view on the future of your organization. Assume (in the coffee shop example) that the vision for the future is that loyal customers only need to pay when leaving the shop. This would imply that you need two different process flows. on top of that, you face some interesting questions: what is a loyal customer, how do you become one, can you loose that status, how do you follow up

Many answers to these questions will be modelled in the fourth submodel, the Action model. Here you describe the business rules. For example, you could define a loyal customer as a customer that came four weeks in a row at least once per week. Several scholars have defined a formal language to describe the business rules. I tend to write them down in natural language whenever encountered during the discussions about the other three models. The public with whom we build the models are (most of the time) not the people that will read formal looking definitions.

1.2.3 Can you create software systems from a DEMO model?

Maybe you can. Several scholars try to do this. Some propose to extend the "essential model" with lower level transactions to include e.g. document creation and the document flow. My impression is that when doing this they are trying to reinvent the hot water while "watering down" the strengths of DEMO.

The goal of DEMO is to model the (intended) operation of a very complex organization on a single sheet. Using DEMO you model the "essence" at a level that can be discussed with C-level and top management. Because of the perceived simplicity of the model, you can have in depth discussions about the intended operation of the organization without going into too much detail. It is of course the task of the analyst to ensure that the model expresses the relevant topics that require further discussion.

Like in every approach to define detailed requirements, you can continue to detail the DEMO model to a level that is so detailed that one can, indeed, create software out of the model "directly" - if of course you define a target software architecture first. My personal opinion is that there are better

approaches (like the one described in this document). One reason is that the detailed DEMO model will most probably (being polite) not result in an evolvable and maintainable system - unless of course you base it on the e.g. the principles of normalized systems theory.

A second observation is that you can probably extend the model with sufficient detail for a small "demo" like problem. These problems are typically sized for inclusion in a book or an academic paper. They do not have the size and complexity of real world problem. Doing this, you completely ignore the strength of DEMO: create a model for a real life complex problem. In a real organization you always need to look at the existing application landscape, integrate with existing application and decide which part of the "model" will need to land on a new, to be implemented, software system. And when you create the DEMO model you definitely do not care about all the details you need to create a system.

It is therefore my opinion that the question, whether you can create a software system directly from a DEMO model is purely academic and hence irrelevant for practitioners.

1.3 MERODE

1.3.1 What is MERODE?

As stated on merode.econ.kuleuven.be, MERODE is an "object-oriented Enterprise Information Systems Engineering method" developed at KU Leuven. MERODE is an acronym for **M**odel-driven **E**ntity **R**elationship **O**bject-oriented **D**evelopment. The most comprehensible introduction to MERODE is the book by Prof. Snoeck [5].

The MERODE methodology was originally developed in the nineties, but has been evolved and elaborated through both scientific research and application in real world cases. Like most methodologies from that era, including the DEMO methodology previously discussed, it has its roots in requirements engineering. MERODE does not focus on gathering the requirements themselves, but rather on the translation of the requirements into a stable and evolvable software system. As it uses a model-driven engineering approach, this translation can be taken rather literally. From the requirements, a number of models are derived. These models are mapped, in a predictable way, onto a predefined software architecture. MERODE is therefore ultimately suited to be combined with a software generation approach.

On the website mentioned above, reference is made to the Merlin tool. This tool can be used to generate software from the MERODE models. I am not aware whether the Merlin tool is sufficiently elaborate to generate a system that can be used by a large organization to support complex requirements or whether it is more of a lab-oriented tool created for educational purposes. It does however demonstrate that the MERODE models can be used for the generation of software.

An example of the combination of MERODE with software generation in a real life, complex environment is described in [6]. In 2000, a group of Belgians founded a telecom operator in the Netherlands (Novaxess). To run a telecom operator you need at least a billing system (Kenan), a CRM system (Clarify), a finance system (Oracle Finance) a network provisioning system and a network management system (MetaSolv). Each of these systems is a huge, complex software system on its own, requiring a complex implementation project. The remaining question is how to integrate these systems without customizing. Customization leads to high maintenance cost and most of the time a long and expensive integration project. One of the major problems for the integration is that the information models of the different systems are partly overlapping but not aligned. An example is the notion of the customer: who is the master, the billing system or the CRM system. Or maybe the network system because that is the service you actually provide. The Architectural vision of Novaxess was to create a so-called integration layer. Every system plugged in on this layer and only communicated with this layer. The process orchestration over the different systems was done by this integration layer. With the support of Prof. Snoeck, a MERODE model was created. From these models software was generated using the Versata system (2000/2001). About 97% of the lines of code were generated. Only a number of complex management screens had to be created manually. In less than one year, the individual systems were implemented as well as the integration layer. This should be compared to the 3+ years it took telenet to implement these systems with a similar scope.

Several years later, the model was reused for a French telematics company. This company offered services like stolen vehicle tracking. While this company used similar but different software components, the project was delivered in about 6 months. Only minor changes to the MERODE model had to be made. The largest part of the model could be reused. Changes were mainly caused because of the differences between a telematics business and a full service telco business. These did not impact the core MERODE model, supporting MERODE's claim of stability and evolvability.

1.3.2 What makes MERODE different?

There are many methodologies that follow the approach of model based engineering. The approach used by MERODE to create a model and transform this into a software system (or at least the core of a software system) is definitely not unique. What makes MERODE different is that at its core it uses a different approach to information modelling. Other methodologies use modelling approaches based on UML class diagrams, entity relationship diagram, object relational models, ... In all these information models there are multiple types of relationships between classes/entities/objects - each relationship type has its own semantics and implied constraints on the object the constraint is linked to. An obvious example is the difference between a 1:n and a 1:1 constraint. In the case of a UML model, you can have an aggregation relation and a composition relation, each of which has a very specific meaning and on the possible instantiations.

MERODE uses another type of information model, with only a single type of relation: "existence dependency". As will be discussed, this is a very powerful concept as it organizes the concepts in the domain in a very natural and stable way. MERODE has named its information model an **Existence-Dependency Graph (EDG)**. It should be noted that, in line with the MDE approach, MERODE does not presuppose an implementation method. The existence dependency relation is between "real world objects". These can be implemented as in-memory objects, persisted in a relational database or in any other way that fits in the targeted (software) architecture.

Because there is only a single type of relation in an EDG, the semantics are clear and simple. The processes that run on top of the information model can be plugged in in a very structured and predictable way. Because of the characteristics of the existence dependency relation, in a MERODE model these processes can only be plugged in at very specific places and are not scattered all over the software system. The attentive reader of the section regarding Normalized systems may see the relation with the so-called ripple effect from NST.

1.3.2.1 Business object (types)

The core of the MERODE (domain) model and of the enterprise model are *Business Objects types*. Business objects are "things" that exist in the reality of the problem domain, though (of course) do not need to be of a physical nature. They are related to concepts that exist in the real world. The business object types represent the concepts that the business people talk about and understand. Technical objects e.g. something you need to display information on a screen are not business objects and will not be included in the enterprise model, while they may be essential for the correct functioning of a software system.

Business Events are "things" that happen in the real world and that impact on or more business objects. In the case of a webshop, the business object could be the Order, the business event could be placing the order. Another event that impacts the Order business object could be the payment, moving its *state* from Ordered to Paid.

The business objects types must have the following characteristics:

- To repeat what is stated above (because it is fundamental), a business object type must have a counterpart in the real world
- Every business object must have an *identity*, even though this can be implicit. This means it must be possible to distinguish between the different instances of a business object type. A person can be identified by e.g. a passport number or a social security number; an invoice by the invoice number ... The identity of business object should not be confused with the (primary) key in a relational database. The latter is, in essence, a technical characteristic and not a property that belongs to the real world. Of course, it is possible to use real world properties as (primary) key in a database.

- They must have a *state* described by one or more attributes.
- They must show some *behavior*

MERODE adds the following requirements to these rather generic ones:

- Every business object must exist for a period of time; if it does not, it is a business event instead. A
- Based on the previous requirement, the life of a business object will start at a certain moment and will end at a later moment. The life of a business object is thus encapsulated between two business events: a *create* event and an *end* event. In between those two events, there can be of course multiple other events. Some of these may change the state, some may not.

Note that, whether something is a characteristic of a business object or is a business object in its own right, depends on the domain you are modelling. For example if you create a model for a shop selling paint or a model for the factory that produces the paint, you will, most probably end up with a difference in the model of "paint".

In the *Existence Dependency Graph* the business objects are related if and only if an instance of a child object exist can only exist if a specific instance of the parent object type exists. This may seem rather abstract, but it is rather simple to understand which I hope will become clear in the discussion of the IES case. The case deals with the transports of goods in containers with a ship.

If we use the concept of existence dependency we can observe that container must exist before you can stow goods in it. If the container sinks to the bottom of the sea, its content will sink as well. The goods are the "container content" and the container content cannot exist without the individual goods as well as the container exists. On the other hand, the existence of the container does not depend on the goods it might carry one day. In a normal situation, the container will be unloaded (instead of sinking), ending the life of the container content but not the life of the container. This is the typical reasoning we make to detect existence dependency. It is a discussion you can easily have with the domain experts, often leading to interesting discussions that require the introduction of new intermediate objects. Using the example of container content: the life of the physical container and that of the physical ship are independent. The container with its content is shipped by land transport to the harbor, put on the ship and at its destination shipped again by land transport. The life of the container content therefore does not depend on the life of the ship. If the container is unloaded at its destination and while it is waiting for the land transport, the container content still exists, but the ship does not anymore

1.3.2.2 Business Events

While the business objects describe the static structure of "what we talk about", the business events are the things that happen in the real world and that impact the business objects. If we return to the example of the bakery, described in the previous section, every step in the DEMO business transaction can be regarded as a business event. This becomes more clear if we would have an online bakery shop instead of a physical one. In that case the customer would put a bread in its shopping cart (the request). This is an event that acts on the object that represents the bread (e.g. by updating the number of breads in stock). It would also create an order for the bread that is linked to me as a customer. It is clear that a business event can impact multiple objects. A stable (correct) state is only achieved if the impact on all objects has been processed. For example, if the order for the customer is created, but there is no update of the stock level, promises (to deliver) will be made without being able to keep these promises.

In some modern systems like in SAP S4/Hana, based on the SAP Hana in-memory database, the stock levels would not be updated but recalculated every time needed, requiring a lot of computation power. This is a design decision based on technological capabilities, which, in my opinion, should

not be taken during business requirements gathering.

In order to be identified as a (potential) business event in MERODE the following requirements need to be satisfied:

- The business event must happen in the real world and must be related to the universe of discourse even when there is no information system.
- The business event must occur at one point in time; its duration is not relevant. For example, incoming payments appear on a bank statement when the payment was made. The fact that it took the counterparty time to make the payment is not relevant unless of course if we are building payment software.
- A business event cannot be decomposed in sub-events.

1.3.2.3 State machines

In MERODE, the lifecycle of a business object type is modelled by means of a *finite state machine*. Business Events trigger *state transitions*. State machines are a well known and widely used concept in Computer science. An introduction to the concepts can be found in this paper [7]. Figure 1.1 is an example of a state machine taken from the discussion of the IES case, later in this document. It shows the lifecycle of a sales order (an order made by a customer). This is a very simple state machine without "loops". Every business event moves the order to a next state.

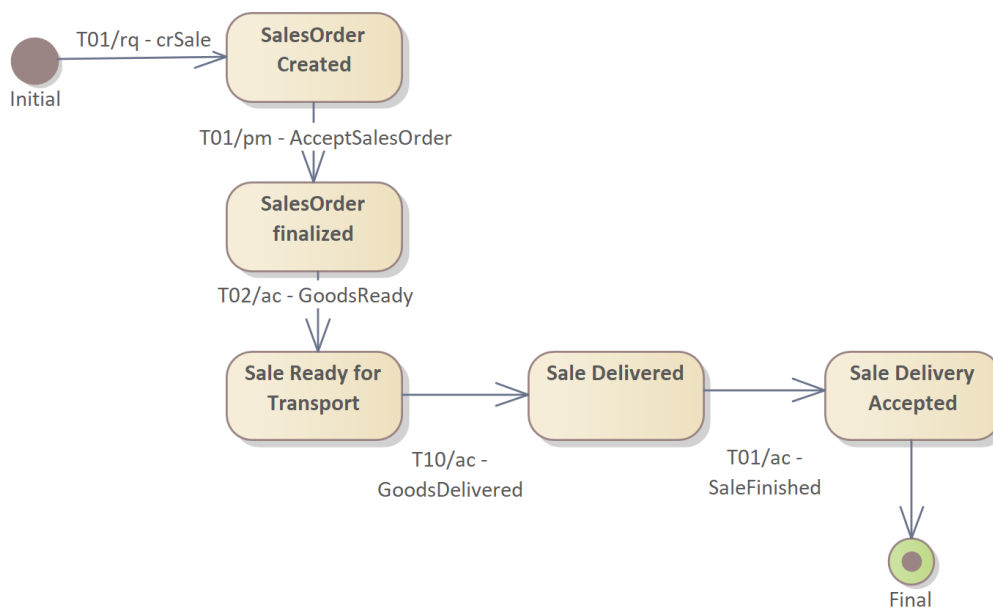


Figure 1.1: State machines for an order

1.4 Normalized Systems (Theory)

1.4.1 Basics

As stated on the www.normalizedsystems.org, Normalized Systems Theory (NST) is a framework that focuses on creating modular structures that are highly evolvable, flexible, and sustainable. It aims to improve the design and engineering of software and other systems to better adapt to constant change and reduce maintenance costs.

In a narrow sense - the one used in this discussion - Normalized Systems Theory (NST) is a *software engineering theory* that answers the fundamental question how to build evolvable and maintainable software systems.

The starting point for NST was the fundamental question whether these evolvable and maintainable systems can be built at all. If it can, it would contradict the the observation expressed in the **law of Lehman**. This heuristic law can be summarized as follows: "The marginal cost of a software change increases every time a change is made". In other words, the cost to maintain software systems increases over time.

There are two aspects to Lehman's law that needs to be discussed:

- what is software maintenance
- what is this cost

1.4.2 What is meant with software maintenance

Paul D. Stachour gives a very good definition in "You don't know jack about Software Maintenance (2009)" and in "Observations about Software Maintenance" (2004). As he states "In software, adding a six-lane automobile expressway to a railroad bridge is considered maintenance—and it would be particularly valuable if you could do it without stopping the train traffic". In all other trades, maintenance is related to the "return of the item to its original state"; in Software, maintenance is concerned with moving an item away from its original state".

Changes are nearly always made under time and budget pressure. If there are three solutions to implement a change, say a good, an acceptable and a cheap, most of the time the cheap one is chosen. The implementation of the change often does not comply with the initial architecture, introduces additional links, introduces unwanted couplings If you run systems that are 20 or 30 years old (like e.g. many core banking systems), the initial structure is deteriorated (at least). For this type of systems, you often have the additional problem that there is nobody left that still knows the internals of the system and that the new developers build "around the old core", deteriorating the structure even more.

This effect is also recognized in the agile approach of software development and the solution was given a name: refactoring. "Refactoring is "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior" (Martin Fowler). In an agile approach one incrementally builds the system in short sprints. This will result in poor code quality, mainly because the developers did not yet have a view on all requirements so could not make the correct judgement calls in the beginning. Refactoring helps to clean up the code and improve the design without changing the behavior. While refactoring is a necessity in an agile development environment, it is often neglected because of time constraints and budget (it works, right - why spend budget on this).

There clearly are many different types of software maintenance, a classification (without intention to be complete) is the following:

1. bug fixes, i.e. error-correction and resolving inadequacies.

2. maintenance related to the support of changed and most of the time extended user requirements.
3. refactoring, cleaning up the code and software architecture without change the (external) behavior
4. changing the technologies and frameworks used in the software system. Some examples of the latter are moving from Oracle DB to PostgreSQL, migrating an application from a windows server only to a linux based installation, ...

Normalized Systems is concerned with the latter two types of maintenance. Why it is not focusing on the first two should become clear in the remainder of this chapter (spoiler: code expansion)

1.4.2.1 The cost of software maintenance

NST is mainly concerned with the two types of maintenance that compare to adding lanes to a highway. We all know that "you can't bake a cake without breaking eggs". Every change has a cost related to it. If you have to add a new user requirement, it *will* cost you, if you need to change the framework used in a software system, it *will* most probably cost you a lot. NST does not try to determine the exact cost of such a change, but reasons from a theoretical point of view about what could make a simple change more or less expensive (relatively). The answer is the so-called *ripple effect* or *combinatorial effect*.

Let us take a real case that has had big impact in the past. When python went from version 2 to version 3, the print statement `print x` to the print function `print(x)`. If you used `print` in all your programs (instead e.g. `logging`), you had to go through all your programs and add the brackets. A small and quick change, which could still take up quite some time if you had a lot of python programs. If you would have created your own `print` function, encapsulating the python `print` statement you only had to make the change once. Many arguments can be made against this example, because this was a change in the programming language. This example only serves at explaining the ripple effect. If the impact of a change "ripples through" your software system, the change will cost you more than when the functionality to be changed was cleanly encapsulated in a single function. To relate this to Lehman's Law: every change will deteriorate the structure (a bit), introduce new couplings in the system and ripple them further through the system. Only by eliminating these couplings, it is possible to create truly maintainable and evolvable software systems.

Can we determine the actual cost of a change using the NST approach? No, but that is not the goal of NST. NST focuses on the drivers that will increase the cost of maintenance and is not an approach to estimate the cost. It basically guarantees that if you adhere to the theory a "a bounded set of changes will have a bounded impact on the system".

1.4.2.2 Can NST beat Lehman?

It will not come as a surprise that it is indeed possible to build software systems that do not suffer from Lehman's law. If the answer were negative, the research would have delivered an interesting academic paper explaining or even proving the law. But on the contrary, NST has formulated four proven principles that need to be adhered to create evolvable software. These are principles are necessary conditions (they must be adhered to), but it has not been proven that they are sufficient or the only principles that exist.

Adhering to the NST principles (described below) results in a very fine-grained modular structure. From the onset, it was realized that it is not possible to create and maintain these high levels of modularity (at a reasonable cost) if a programmer needs to write the code by hand. Therefore, in parallel with the development of the theory, so-called "software expanders" were created. These expanders are used by the "NSX", a spin-off from the University of Antwerp that develops and

maintains the expanders and uses them to build real systems. A selection of user stories can be found on the website from the NSX www.normalizedsystems.org.

These user stories prove that it is possible to build working and performant software systems using the NSX expanders based on NST. Note that this not prove that the systems build are truly evolvable. Adding new user requirements can be achieved by adding the functionality to the input of the expanders and regenerate the system. If done right you should have a new system that supports the additional requirements. This "if done right" requires, at least, that the data that was already in the initial version of the system can still be used in the new system. Needless to say that one of the NST principles deals with this issue.

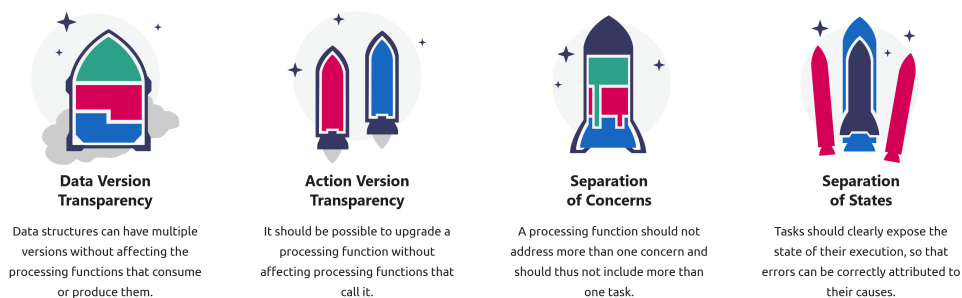
The user stories and the experience build up in the NSX only proves that software generation and/or expansion using the expanders based on NST *works* for real systems (complex, lots of data) and results in performant systems.

This also holds for other software expanders. Nowadays, for code generation often "low code" systems are used. Starting from visual drag-and-drop building blocks and pull-down menu interfaces, often extended with the addition of business rules (conditions) and "code snippets", the code is automatically generated by these low-code systems. These systems are often used for automating workflows. A well known example is Salesforce's low code development platform that allow users to configure and customize the application. The main difference with Normalized Systems is that these low code systems all require an expensive engine to run upon. This engine needs to be maintained by the vendor. As any piece of software that was not developed with "evolvability" in mind, the maintenance cost will increase and the cost for using the engine will most probably go up. On top of that, you are limited to the frameworks and e.g. security components that the vendor is willing to support.

The Normalized Systems expanders on the other hand generate code that can be compiled with a standard (java) compiler. An experienced user-developer can create his own expanders or adapt existing expanders to incorporate e.g. the use of a specific security component that is required in his environment. I guess the help of one of the specialists of the NSX will be required in the beginning.

1.4.3 Normalized System Principles and expanders

Normalized systems theory formulates four basic principles for evolvable software. You must adhere to these if you want to build evolvable software. It has however not been proven that adhering to these principles is sufficient. I personally think it is not, one of the issues NST does not deal with is the issue of data migration. I argue that the core of MERODE, being the Existence Dependency Graph and the related "object event table" (not yet discussed) are an extension of NST required to deal with this issue. This is however not part of this document



Based on these principles, the expanders have been created. The software architecture is described in detail in the Normalized Systems White Paper "Model-Driven Software Engineering: An NST

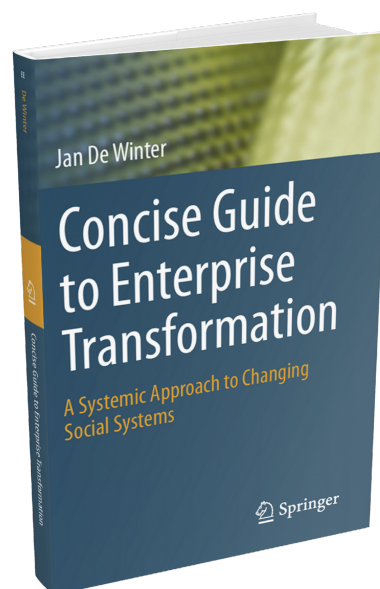
Architecture" by Prof. Herwig Mannaert. The interested reader will notice that the architecture is very close to the MERODE architecture. The MERODE models can be used as an input for the NS expanders. An important requirement is to assure that a business event is consistently executed on all the objects the event is involved in. Remember from the example in the MERODE section that when ordering a bread, the bread object has to be updated to reflect the correct stock level. If the impact is not consistently executed on all relevant objects, the "model" will be in an inconsistent state. In the solution of the IES case, this topic will be discussed in more detail.

1.5 Things to read

The methodologies discussed are all rooted in scientific research and are taught in multiple academic courses. As such they are well documented in books and research papers. The literature mentioned below is a selection based on my personal preference.

1.5.1 Appreciative Inquiry and Enterprise Engineering

The starting point ...



1.5.2 Enterprise Engineering

The DEMO methodology is owned by the Dutch *Enterprise Engineering Institute*. Their website www.ee-institute.org links to most literature available pertaining to this subject matter. You can also find several cases studies and clarifying introduction.

A tool to draw DEMO models can be found on www.yadm.info, This is a new Belgian initiative that will eventually include a practitioner oriented online course.

If you have time to invest and want to acquire a deep knowledge of enterprise engineering and DEMO, the books in Springer's Enterprise Engineering Series are probably the best resource.

The starting point are the 3 books by Jan Diets and Hans Mulder.



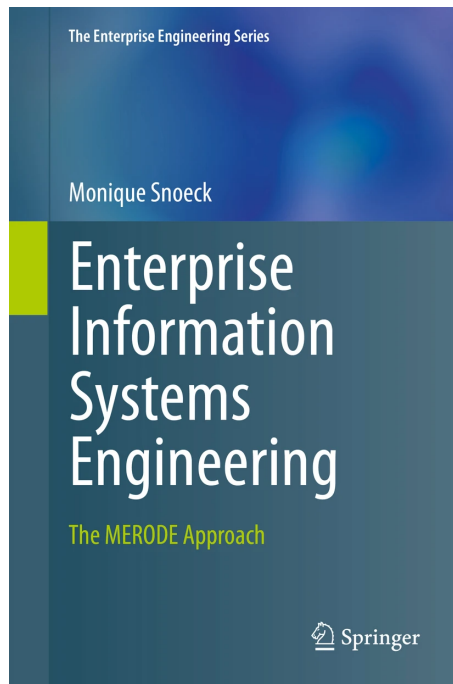
Enterprise Engineering is about the design of organizations, and *not* about software design. The books of Jan Hoogervorst focus on the link between enterprise engineering and enterprise governance.



1.5.3 MERODE

An overview of the information regarding the MERODE methodology can be found on the website www.merode.econ.kuleuven.be.

A complete and very comprehensible description of the methodology and the underlying theory can be found in the Merode book that is part of Springer's Enterprise Engineering Series.

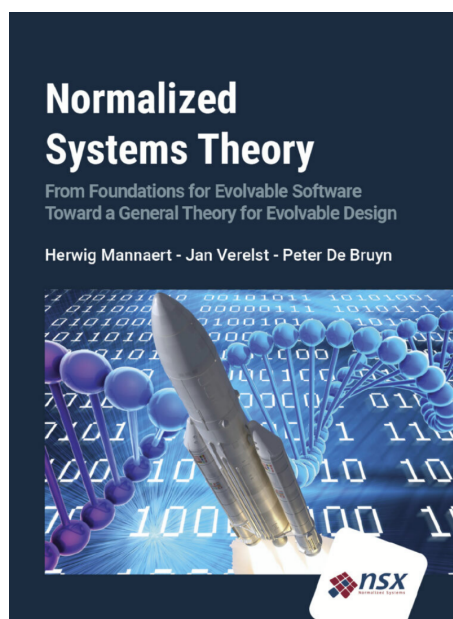


The website also includes links to the course material that is used at KU Leuven.

1.5.4 Normalized Systems Theory

A lot of information regarding Normalized Systems Theory can be found on this website www.normalizedsystems.org. Especially the six lectures on www.normalizedsystems.org/academy can be advised.

The theoretic foundation of the theory is described in a text book used at the University of Antwerp.



For most of the detailed theoretic discussions, especially the extensions of the theory into other domains like process design, enterprise design, . . . , you need to read (study) the academic publications.

2. Case Study IES

2.1 Introduction

In this chapter, we discuss a case with sufficient complexity to demonstrate the concepts that have been discussed in the previous chapter. We will start with the DEMO model and expand it into a consistent MERODE model.

The case that has been selected is the **IES case**, the Import-Export Services case. This case was originally created by Prof.Em.Dr. Henk Sol from TU Delft. Later it was used by Prof.Em.Dr. Jan Dietz and Prof.Dr. Hans Mulder to demonstrate how DEMO helps to reduce the complexity of the model. The case is often used in DEMO courses as one of the more advanced cases students have to solve. The IES case is also discussed in detail in the book that explains the theoretical foundations of Enterprise Engineering and that introduces the DEMO methodology [2]. In this book, it is called the *GloLog* case. Interested readers are referred to this book for a detailed discussion on how to derive the DEMO model (with the 4 submodels) from the textual description, and esp. to understand how the model for a rather complex problem can become that simple. The reasoning will not be repeated in this document, the DEMO models are considered as an input.

The *GloLog* case is introduced in [2] p.385 as follows:

The case *GloLog* is an exercise in producing the essential model of an enterprise that offers the transporting and storing of tangible things: goods, containers, ships. The CM, PM and FM of the *GloLog* organisation are presented and discussed. Emphasis is put on four topics. The first is the unavoidable need to conceive notions that are not present in the narrative description but that are nevertheless ontologically crucial: the container content and the ship content. The second is the necessary existence of four distinct, only 'loosely coupled', business processes. The necessity stems from the incompatibility of the case cycles in these processes: sale, purchase, container content and ship content. The third topic is the distribution of responsibilities: the ontologically necessary ones, as represented in particular by the CM, cannot easily be traced back to the narrative description of the case. The fourth topic is the current implementation of the essential model. Without the help of the essential model, it is almost impossible to clarify the role of the many parties involved and the many document kinds used (respectively to point at their redundancy).

It should be noted that the complexity that exists in the real world is *not* reduced by using DEMO; the complexity does not magically disappear. In DEMO, you model the *essence* of the organization only. You leave out a lot of the complexity such that it becomes easier to reason and discuss the future organization and way of operating. This also implies that if you need to build software systems to underpin this (new) operation, you will need to add a lot of detail. When you start from the DEMO model however, you can do that while remaining consistent with the shared mental model that was established when constructing (and agreeing to) the model [1].

It should also be noted that the case covers a large part of the operations of a real shipment company. It cannot reasonably be expected that we can build a detailed model and generate an IT solution for the entirety of this operation in a case study. We will therefore carve out a part of the model and detail it further in a MERODE model. The area we have chosen is related to ordering and invoicing. The reason we have chosen this part is (1) because it will clearly demonstrate that the DEMO model does not contain all the detail we need, (2) because the order-to-cash process (ordering/invoicing/paying) is a well known process that requires little explanation and (3) because a detailed MERODE case is available for this part of the process.

This (already existing) MERODE case is not related to an import-export shipment business, the domain model and the reasoning can however be (re)used, demonstrating the power domain models,

esp. models based on existence dependency. A (correct) existence dependency model does not change if you apply it in another domain. It can be argued that the fact that the existence of a "thing" that exists in the real world (tangible or intangible) depends on another "real thing" is a universal characteristic rather than a modeling choice.

The original case description is in Dutch. The text has been translated automatically into English. Both documents are provided in the next section.

2.2 Case Description

2.2.1 Terminology

Before laying out the case description, some technical terms are defined in this section. The translations of these technical terms has been aligned (in most cases) with the GloLog case.

The following table shows the translations from the original Dutch text to the English version.

Dutch	English	Definition
douane cargadoor	customs shipbroker	Shipbrokers are specialist intermediaries/negotiators (i.e. brokers) between shipowners and charterers who use ships to transport cargo, or between buyers and sellers of vessels.
terminal-kantoor expediteur verlader	terminal office freight forwarder , shipping agent shipper	
rederij stuwadoor	shipping company stevedore	The party that wants to have the goods shipped. This is often the party that produces or delivers the goods, but can also be the party that will receive the goods. In the GloLog case this is translated as the supplier the company that Stevedores are workers who oversee the loading and offloading of cargo at international ports.
landvervoerder	land transporter	

Demurrage in Shipping (copied from gofreight.com)

Demurrage is a charge levied by the ocean carrier (shipping line) when a full import container remains at the port terminal beyond the allotted free time after the vessel has discharged. .

In simpler terms: once the carrier unloads your container from the ship and places it at the terminal, you have a set number of free days to pick it up. If you don't pick it up within those free days, demurrage charges begin accruing on a per-day, per-container basis.

Demurrage is essentially a penalty for using the port terminal as "free" storage

2.2.2 The organization

The organization Import/Export Services (IES) is a company based in Nieuwegein that manages the entire logistics chain for its clients. One of its tasks is importing consumer electronics products from the Far East for European dealer companies. This involves both sales companies and business groups. The products are transported in containers by ship. IES coordinates the entire process from ordering to delivery of goods to the client, using services from various other organizations.

To ensure smooth container throughput, not only must the physical transport actions be carried out quickly and accurately, but also the information-intensive processes surrounding them. A good mutual coordination of the involved organizations is essential. When importing containers by ship from the Far East to Europe, several organizations are involved alongside IES. Figure 1 shows these organizations in their mutual relationship. The physical flow of goods runs from the shipper via the

shipping company (the ship), stevedore (container transshipment), land carrier (transport Rotterdam – Nieuwegein) to IES, which takes care of temporary storage and final delivery to the client. The cargo agent and freight forwarder play a coordinating role, while customs has an authorizing role. The latter three organizations also have employees on the port premises, housed in the terminal office.

Between the various organizations, information is frequently exchanged by e-mail, phone, fax, and courier to get the goods to their destination as quickly as possible. In the years leading up to the time this case takes place, the volume of trade has increased significantly: currently about 4000 containers (approximately 800,000 cubic meters), a doubling compared to the volume three years prior. However, the competitive pressure on IES is high. The company is therefore considering measures that will increase the quality of its logistics services and/or (preferably and) reduce costs. The expectation is that a reduction in throughput time will lead to a decrease in costs, particularly because then less demurrage needs to be paid. Additionally, it is expected that redesigning and restructuring the current business processes will lead to lower processing costs, especially in terms of time spent per container.

2.2.3 General outline of the import process

The execution process can be briefly described as follows: IES processes orders from its clients (dealer organizations) and orders the requested products from a supplier in the Far East. This supplier (the shipper) assembles the requested shipment and places it in one or more containers on a ship from a shipping company. This ship arrives in Rotterdam where it is unloaded by the stevedore. Subsequently, the containers are picked up by trucks from the land carrier and brought to IES. IES unloads the container, after which the truck takes the empty container back. The individual orders are finally delivered by IES's ordering service to the dealer organizations.

It requires considerable care to ensure this process runs smoothly. If we limit ourselves to the actions in Europe, the following activities and responsibilities are involved:

- IES initiates the shipments and verifies (and corrects if necessary) the freight documents sent by the shipper, forwards these to the freight forwarder, receives the goods, and ultimately delivers them to its clients. IES plans the container transport and informs the freight forwarder when and where which containers need to be picked up.
- The cargo agent acts as the shipping company's agent and therefore has authority over the ship and cargo. The cargo agent must provide customs, IES, and the freight forwarder with information about the cargo and also communicate the expected arrival times of a ship as well as the times when containers are unloaded (after which they are available for further transport).
- The stevedore performs the container transshipment and must make arrangements with the shipping company (the ship) about where and when the containers will be unloaded. The stevedore stores the containers on its own premises until they are picked up by trucks from the land carrier.
- The freight forwarder takes care of the customs clearance of the containers and the subsequent release of this particular part of the cargo. The freight forwarder then arranges, in accordance with the planning of IES, the container transport from the stevedore's premises to the establishment of IES. To this end, he makes arrangements with the land carrier about which container can be picked up at what time and provides the land carrier with the documents needed for picking up the containers at the stevedore.
- The terminal office houses the offices of customs and various cargo handlers and freight forwarders on the port premises.

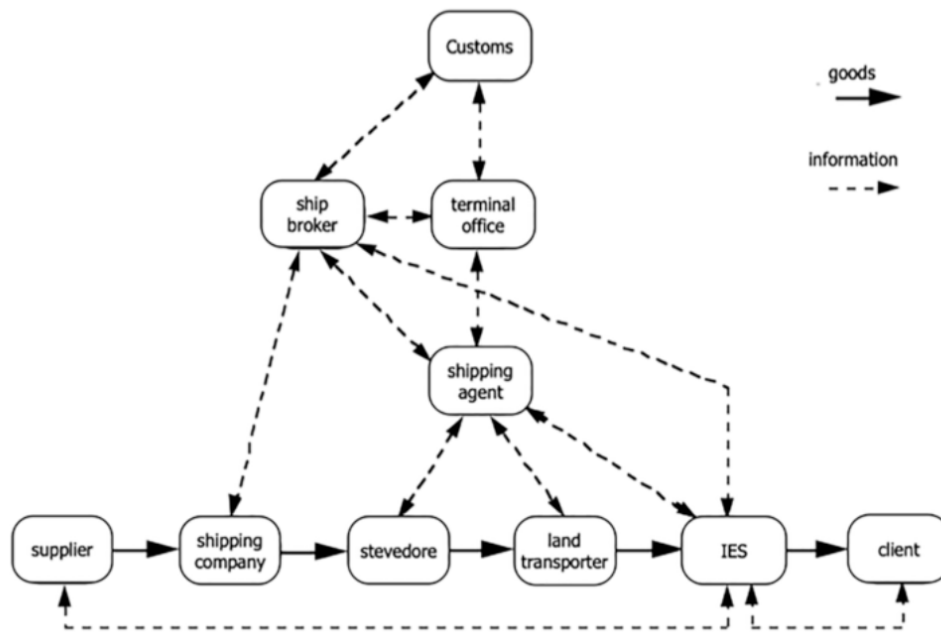


Figure 2.1: Global coherence of the involved organizations

2.2.4 Business processes in more detail

IES places an order based on client orders in the Far East. The shipper there prepares a shipment based on this order that can consist of one or more containers. Depending on the port from which it is shipped, the shipper prepares a Waybill (from Singapore) or a Bill of Lading (from Tokyo) for a shipment. A Waybill is a transport document "in name," which means that the recipient named on the document must identify themselves to receive the shipment. A Bill of Lading (BoL) is "to bearer" and thus serves as the official proof of ownership of the shipment. A BoL is therefore negotiable and often worth a lot of money. Both transport documents specify which containers are involved, what exactly is inside, who the shipper is, who the recipient is, the voyage number, and finally the parties that need to be informed that the relevant containers will arrive at the port. However, the type of transport document used has consequences for the handling process in Rotterdam. Waybills and/or BoL are sent together with other documents (invoices and packing lists) to IES via courier service. This takes about 2 days; sea transport takes more than three weeks. IES administers the incoming BoL, verifies whether they match the placed orders, makes copies, and then forwards the documents to the freight forwarder.

Between two and seven days before the arrival of a ship, the shipbroker receives the manifest from the shipping company with information about the precise cargo of the ship. Each item on board the ship has its own item description and a unique item number in this document. A manifest typically consists of about 2000 pages. Based on the manifest, the shipbroker prepares a general declaration (GV) that is sent to customs. This GV contains a list of all Waybill BoL numbers of shipments that are on board. The shipbroker also prepares so-called "pre-arrival notices" (PaN) that are sent to the parties mentioned in the manifest.

The freight forwarder receives the relevant shipping documents from IES (and also from other organizations that have cargo on the ship), checks them for completeness, and archives them under the name of the respective ship. By the time the ship arrives in the port, the freight forwarder prepares the documents that the truck drivers of the land carrier need to receive a container from the stevedore and drive to the IES location. The following documents are involved:

- For each container, the freight forwarder creates a so-called CMR (Convention relative au contract de transport international de Marchandise par Route), a European freight document that must be shown by a truck driver during land transport.
- If it is a shipment "to bearer" (thus with a BoL), the BoL is exchanged by a courier at the shipbroker for so-called tracking slips. The shipbroker creates a tracking slip for each container listed on the BoL, which must be presented to pick up the container from the stevedore. For shipments "in name," a copy of the Waybill and identification as the carrier is sufficient for the recipient of the shipment.
- For each container, a customs document must be prepared, which must include not only the details about the goods as specified in the Waybill or BoL but also the number of the GV and the item numbers according to that GV.

Getting these documents correct is not always straightforward. The PaN usually (but not always) contains the item number and the number of the GV in addition to the item description. Sometimes the freight forwarder does not receive a PaN at all because they are not listed as a party in the manifest. Besides the PaN, the item number and the number of the GV are generally mentioned on the tracking slips. However, no tracking slips are created for shipments with a Waybill. Therefore, the freight forwarder regularly contacts the shipbroker by phone to obtain the necessary information for the customs documents.

Customs checks whether the specified data on the customs documents provided by the freight forwarder matches the GV provided by the shipbroker and inspects, if deemed necessary, the actual contents of the containers. If the data does not match exactly, the freight forwarder must provide clarification. If everything is found to be in order, customs sends a release notice back to the freight forwarder along with the stamped customs documents. The freight forwarder, in turn, informs IES which containers have been released by customs. IES determines when the land transport of the released containers can best take place, taking into account the urgency of the various shipments and the available unloading capacity at its own location, and communicates the resulting schedule to the freight forwarder.

Based on IES's schedule, the freight forwarder issues transport orders to the land carrier. These orders are confirmed by fax, with a copy faxed to the terminal office. A truck driver submits the confirmation fax at the freight forwarder's terminal office and receives – provided that the fax matches the copy at the terminal office – the relevant customs documents, CMR, and Waybill or tracking slip. The stevedore loads the relevant container onto the truck upon presentation of the CMR and Waybill or tracking slip, after which the driver drives to the IES location. IES receives the container and the accompanying documents, checks the shipments for completeness and possible damage, temporarily stores them if necessary, and ultimately delivers them with its own delivery service to the respective dealer organizations.

2.2.5 Current Task Execution

The tasks of IES and the freight forwarder described in the previous section are elaborated here at the level of activities of individual employees. The tasks of the other involved parties do not require further elaboration for the purpose of this case.

At IES in Nieuwegein, three employees (hereinafter referred to as A, B, and C) are involved in receiving and administratively processing regular shipments. Employee A processes the documents that are sent by courier by the shipper in the Far East. The number of sets (BoL or Waybill with packing list, invoice, and other documents) in a courier shipment varies from 1 to 12 (averaging about 3). The sets are sorted and copied, after which the documents that are relevant to the freight forwarder are sent by post in a folder to the head office of the freight forwarder in Rotterdam, along with a cover letter (documents that are ready before 4 PM are delivered the next working day

around 11 AM). The information on the BoL or Waybills is entered into the IES database system. For each shipment, a file is maintained in the "in transit archive" containing the client's order form, the (copies of the) transport documents, and the relevant printout from the database system.

Employee B, in consultation with the Inventory Management Unit of IES, creates the container planning based on the expected arrival times of ships, PaN's, and the release notifications from the freight forwarder. This planning is communicated to the freight forwarder by phone and then confirmed by fax. The scheduled pickup times for the containers are recorded in the relevant files, while a note is made on the planning board. Furthermore, employee B retrieves the files of all containers expected that day from the "in transit archive" daily. Using the database system, a message of arrival (BvB) is created for each container and passed on to employee C.

Employee C looks up the corresponding BvB each time a container is delivered by the land carrier, inspects the shipment, and sends the BvB back to employee B along with the documents provided by the truck driver (and in case of damage or loss, also an incident report). Employee B compares these documents with those in the file (and reports any irregularities to employee A), makes copies of the customs documents, and archives them in the file. Those documents that ultimately need to be delivered together with the shipment to the client are sent with a copy of the order form to the Order Service of IES. The processed container is removed from the planning board.

At the freight forwarder in Rotterdam, two employees are involved (hereinafter referred to as D and E). Employee D performs the actions that precede the release of a container to IES. Employee E performs the actions related to organizing the transport from the port in Rotterdam to IES in Nieuwegein. Employee D checks the documents (BoL or Waybill, packing lists, and invoices) that arrive daily from IES, enters the data into the internal computer system, and archives the documents in the file of the relevant ship. For each container, a CMR is printed by the computer system. In addition, employee D processes the PaN's from the cargo owner. Sometimes the PaN already contains all the information needed to prepare the customs document for the relevant shipment, which is then done immediately.

About two days before the arrival of a ship, employee D inquires with the cargo owner about the availability of the tracking slips. As soon as these can be picked up, the BoL is taken to the cargo owner and exchanged for tracking slips. The delivery of the BoL and the pickup of tracking slips is done by a courier from the freight forwarder who rides to the cargo owner's office twice a day on a moped. Based on the tracking slips, employee D prepares the customs forms as far as they are not yet ready. For shipments with a Waybill (thus without a tracking slip), the cargo owner is called if necessary for the item numbers and the GV number. Finally, the customs documents are taken together with the corresponding tracking slips and CMRs by moped to customs at the terminal office. After approval, customs forwards the documents to the employees of the freight forwarder at the terminal office, where they are later picked up by a driver from the land carrier. Customs notifies the head office of the freight forwarder by telex that a container has been released. Employee D checks several times per day or if there are exemptions received by telex; if that is the case, then a fax message is created and forwarded to IES. Employee E processes the container planning as it is communicated by IES (and also by other companies) to the forwarder. He searches for the relevant files and agrees by phone with the land carrier when a truck will pick up which container. For each container, employee E records the verbal agreement in the form of a transport order. This order is faxed to both the land carrier and to the own employees at the terminal. A truck driver from the land carrier arrives at the terminal site at the agreed time, enters the terminal office, picks up the documents from the forwarder's employees (he only receives them when his fax matches the one that employee E has faxed to the terminal office), drives to the stevedore's site, receives the container there upon presentation of the delivery note or Waybill, and finally delivers it along with the documents to IES in Nieuwegein.

2.2.6 Some Performance Indicators

IES aims to have all containers in-house within 5 working days after the arrival of the ship. The average is currently around 5 days; about one-third of all containers are delivered later (see figure 2.2). As can be inferred from the process descriptions, the "throughput time" in figure 2.2 is related to a large number of factors. To gain insight into the duration of different steps in the process, a number of time intervals have been determined based on a sample of 200 containers, as shown in figures 2.3 to 2.7.

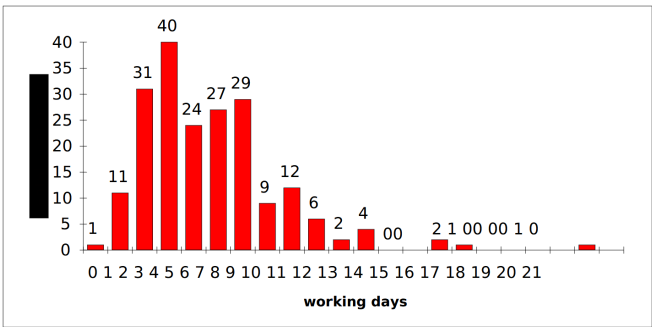


Figure 2.2: Time between arrival of ship and delivery of container delivery

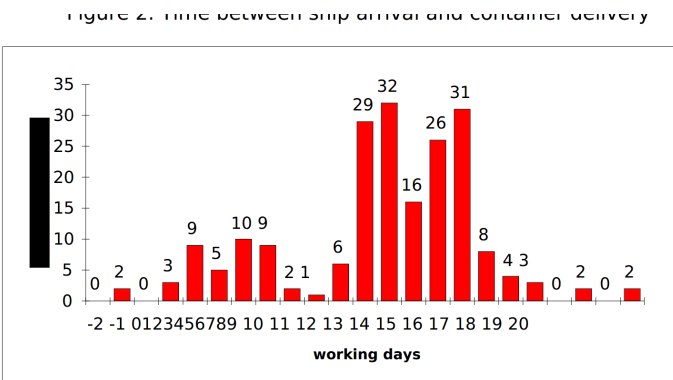


Figure 2.3: Time Between Ship Arrival and Release by Customs

As a standard, IES requires that the transport documents sent by shippers must be received seven working days before the ship’s arrival time. Figure 2.3 shows that this standard is not met for one-fifth of the containers in the sample. For one shipment (2 containers), the documents arrived at IES only one day after the ship’s arrival!

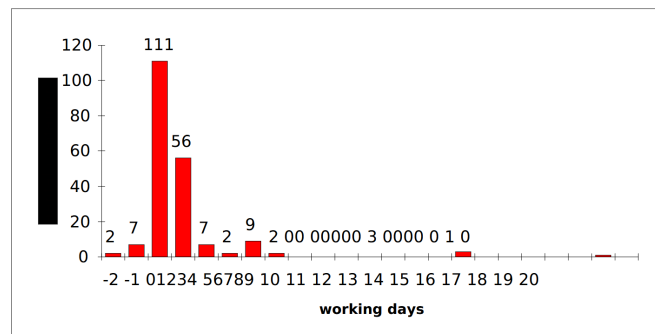


Figure 2.4: Time Between Ship Arrival and Release by Forwarder

Figure 2.4 shows the distribution of the number of working days that elapse between the moment the ship arrives in Rotterdam and the moment the containers for IES on that ship are released by the forwarder. The sample shows that it occurs that customs releases a shipment before the ship has arrived. Approximately 60% of the containers are released before or on the day of arrival. Notably, there are containers that were only released after 13 and 19 working days after arrival. The two respective shipments were severely delayed due to communication problems between the shipping company (represented by the cargo handler) and the forwarder.

To determine the extent to which the timing of release by the forwarder depends on the moment of release by customs, the time between the ship's arrival and release by customs was also determined based on date stamps on the customs documents. This is shown in Figure 2.5.

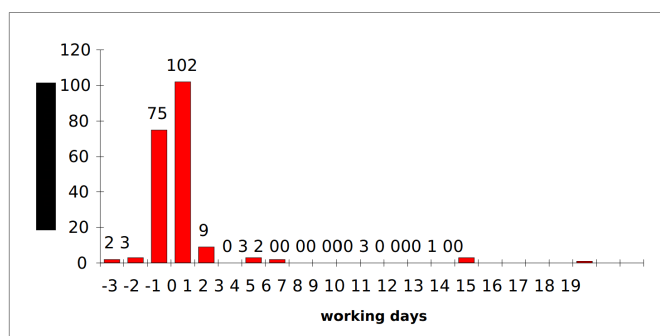


Figure 2.5: Time Between Ship Arrival and Release by Customs

The comparison of Figures 2.4 and 2.5 has led to a further determination of the time difference between release by customs and release by the forwarder. This difference is shown in Figure 2.6.

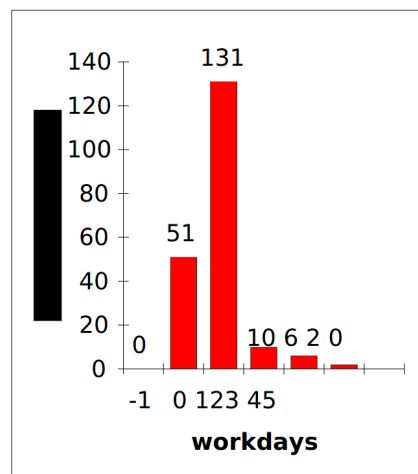


Figure 2.6: Time Between customs release and release by the forwarder

From Figure 2.6 it appears that the release of a quarter of the containers is still passed on by the forwarder to IES on the same day. In fewer than 10 Figure 2.7 shows the difference between the creation date as stated on the customs document and the date on which customs approved the document. The chart therefore provides an indication of the speed at which customs works. It should be noted that the creation date on the customs document does not always match the time of dispatch. It regularly happens that on the day of the creation date the customs document cannot yet be fully completed because the GV's shipper number or the item number still needs to be requested from the cargo holder. The document is then only submitted to customs later; Figure 2.7 thus suggests a longer processing time than is actually the case.

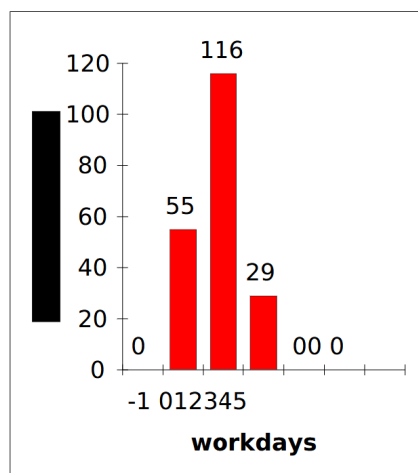


Figure 2.7: Time between creating the customs document and release by customs

As indicated at the beginning of this case description, IES thinks it can save costs by reducing the transit time of containers. The relationship between transit time and costs is most clearly seen when looking at the demurrage charges (demurrage costs): when containers remain at the terminal for longer than 3 workdays, approximately €100 per container per day is charged to IES. A second factor of importance is the interest burden: since it involves fairly expensive goods (the average container load has a value of about half a million euros!), one day of delay at an interest rate of 6% per year amounts to approximately €80 per container. This possible saving, however, benefits

not IES but the shipper. As a third factor, the personnel costs can be noted. It seems obvious that efficiency improvements in business processes lead to savings in working time and thus in personnel costs. Since it is difficult to prove that 25% less work actually leads to 2% fewer personnel costs (it is highly questionable whether a job actually disappears), this factor weighs less heavily than the previous two. Besides these three cost factors on which IES believes it can save, the following costs are worth mentioning:

- sea transport from the Far East to Rotterdam costs about 5000 euros per container;
- truck transport from Rotterdam to IES in Nieuwegein costs about 400 euros per container;
- sending documents by courier from the Far East to IES costs about 150 euro per shipment

2.3 Discussion of the EIS Solution

We start from the DEMO models as presented in the book [2]. As already stated, we strongly recommend to read chapter 18 of this book to understand how this model is derived from the description and why this is an "ontological model" of IES. In this discussion we will start from the DEMO model and we will not derive it. The goal of this discussion is to show that we can integrate the MERODE approach with the DEMO approach and derive the MERODE models that can, in a next step, be used to generate a software system. The detailed MERODE models and esp. how to are impacted by changes in the requirements are not part of the discussion either. For this we refer the reader to the presentation of Prof. Snoeck and to her course notes on orders and invoices.

The first diagram is the Cooperation Model 2.8. We will make abstraction of the detailed (correct) terminology used in the book (officially this is called a CSD) as we do not expect the reader to be a DEMO expert.

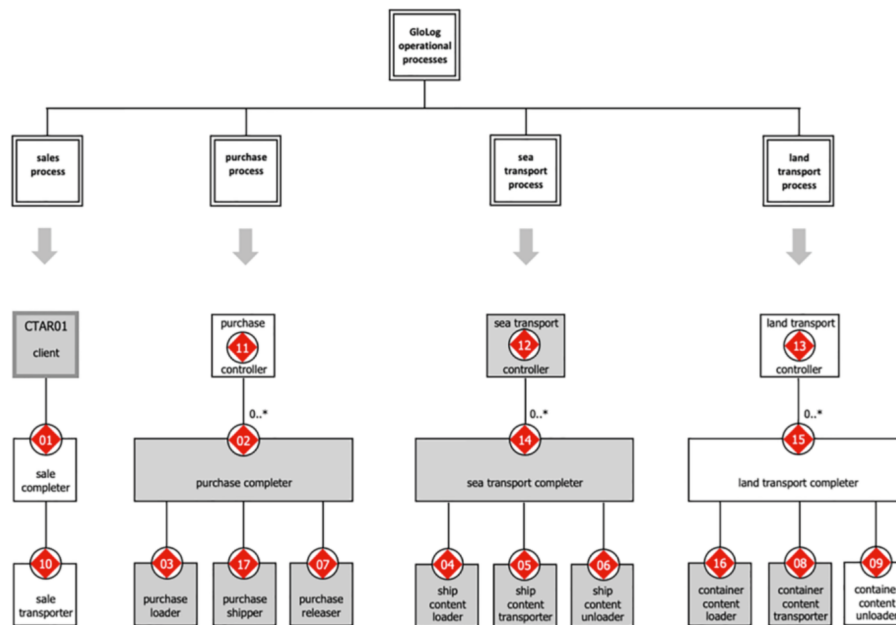


Figure 2.8: CM of the organization

The rectangular boxes in the lower part of the diagram (below the arrows) are *responsibilities* or *roles*. The white boxes are roles within the IES organization; the grey boxes are roles outside the organization.

The red circled diamonds are the *commitments* or *transactions*. A commitment is (normally) between two roles. There is a role that requests something, there is a role that commits to produce (tangible or intangible) what was requested. Every commitment is identified by a number (e.g. 01). The commitment or transaction of 01 is called "T01". This identification is used in the tables below.

The transactions T11, T12 and T13 are red diamonds in a box. These are "self initiating" transactions. For example for T12 there is a role (responsibility) to control on a periodic basis whether a ship can leave. In DEMO, this control is the execution of the task, not the trigger. There is no other role triggering this action. There clearly is a commitment in this transaction: the executor of the task is committed to do the periodic control.

Figure 2.9 shows a single transaction. This is the core transaction of the IES case.

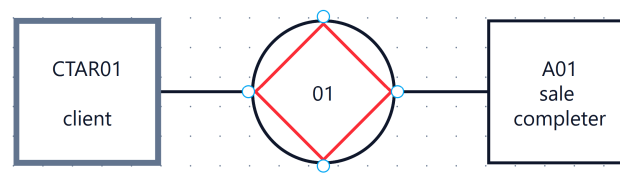


Figure 2.9: The main transaction T01

The transaction is initiated by a client who orders products from IES. It only terminates when the products are delivered to the client. The initial order event (request), followed by IES accepting the order (promise to deliver), triggers the entire process flow described in the model.

Most readers will raise their eyebrows. Isn't there an order, an invoice, a delivery note ... in the model? These are documents that are essential for the operations of the business and even legally required. In the *essential* model of the operation these are not modelled.

The order can be considered as being part of the request/promise part of the transaction T01; an invoice can be considered as being part of the state/accept part of the same transaction.

The reader of the case solution may also notice that the payment transactions are missing (payment by the customer, payment to the supplier). The reader who is familiar with contemporary logistics flows may identify other missing transactions and documents.

It may also be noticed that the case solution does not contain a supplier object (concept). This is because IES is an internal department of a multinational company that delivers HIFI equipment to shops and dealers in the Netherlands. Their only supplier is the company they belong to. The only products they can buy and sell are part of the product catalogue from the mother company. The suppliers are internal production facilities of the group. Internal invoices for the purchase and shipment of the equipment is not part of the IES scope, neither is the payment. One can assume that this is coordinated at group level through some form of intercompany billing. Similarly, it can be assumed that the invoice to the customer and the related payment is handled by another department in the group and that it is not part of the scope of IES.

These assumptions reduce the complexity of the solution. In this document we will first present a solution to the case "as is". In a second step we show for a small part of the solution how the initial model can evolve towards a more generic solution supporting more complex requirements. This will show that the essential model of DEMO needs to be extended slightly to reflect the change. These changes will of course impact the related MERODE models. The impact of the changes will however be limited. They do not cause "ripple effects" through the entire application.

Another important observation about this "essential model" is that it does not have the detail needed for a organization redesign. It does define the essential processes, process steps and responsibilities. It does not yet specify the business rules to be applied. For example, in the past a single order of a customer was placed with a single supplier. IES did group several orders into one to reduce shipment costs. When redesigning the organization, we would need a management decision that this remains the business model or that, in the new way of working they might split a customer order and allocate it to different suppliers.

In the current way of working, the supplier is responsible for initiating the shipment. This may be optimized further by using an independent freight forwarder and group the orders together to reduce the number of containers. All these decisions need to be added to the model. This will, however, not change the essential model fundamentally.

Four processes have been identified in the case:

- The sales process
- The purchase process
- The sea transport
- The land transport

Each process is represented in the CM as a tree of transactions. Each of these trees seem to exist independent. People that have experience drawing process models (e.g. using BPMN) probably expected an end to end workflow starting from the sale all the way to the delivery. This is because in the CM the flow of commitments is modelled, and not the flow of the production.

The link between the trees can (and should) however be modeled in the CM. This complexity was not represented in 2.8. The full model, including these links, is represented in 2.10. Two types of links have been added:

- interstriction relations (dashed lines)
- interimpediment relations (dotted lines with arrow)

The interstrictions link a transaction with an actor stating that the actor requires information about the transaction. For example, the purchase controller must know about what has been ordered by the clients before he can place an order with a supplier. When we draw the process model we must carefully analyze which state of the transactions must be linked. If we link the "accept" of transaction 01, which occurs when the client accepts that all the goods have been delivered, we get a deadlock.

The interimpediment relations is a wait link between actor roles. It expresses that one actor can only start (or continue) if another transaction it is not directly related to has reached a certain state.

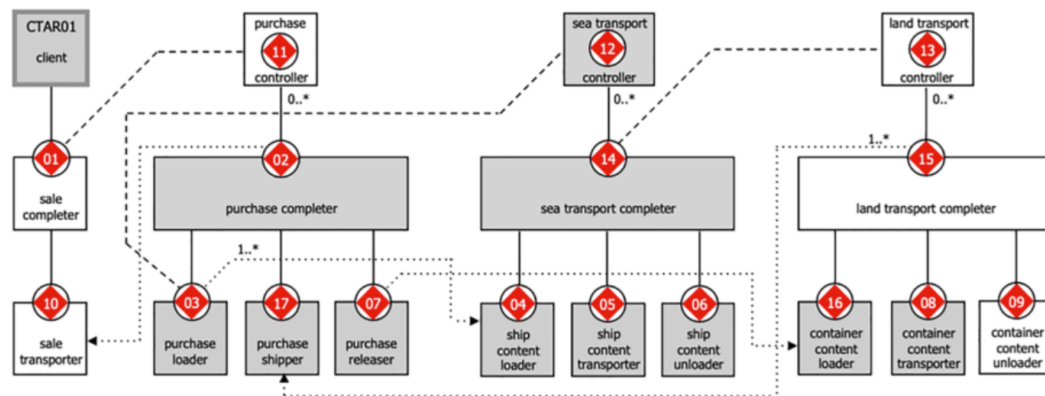


Figure 2.10: Full CM for IES

2.3.0.1 Building the Fact Model

We start with the Fact Model that was part of the official solution of the case in 2008 (DEMO version 2), as presented in figure 2.11. In DEMO 2, the syntax of the FM was still closely related to the syntax ORM (Object Relational Modeling). In the current version DEMO-4, the syntax of the fact model has moved away from ORM. The DEMO-4 model is taken from p409 in[2]

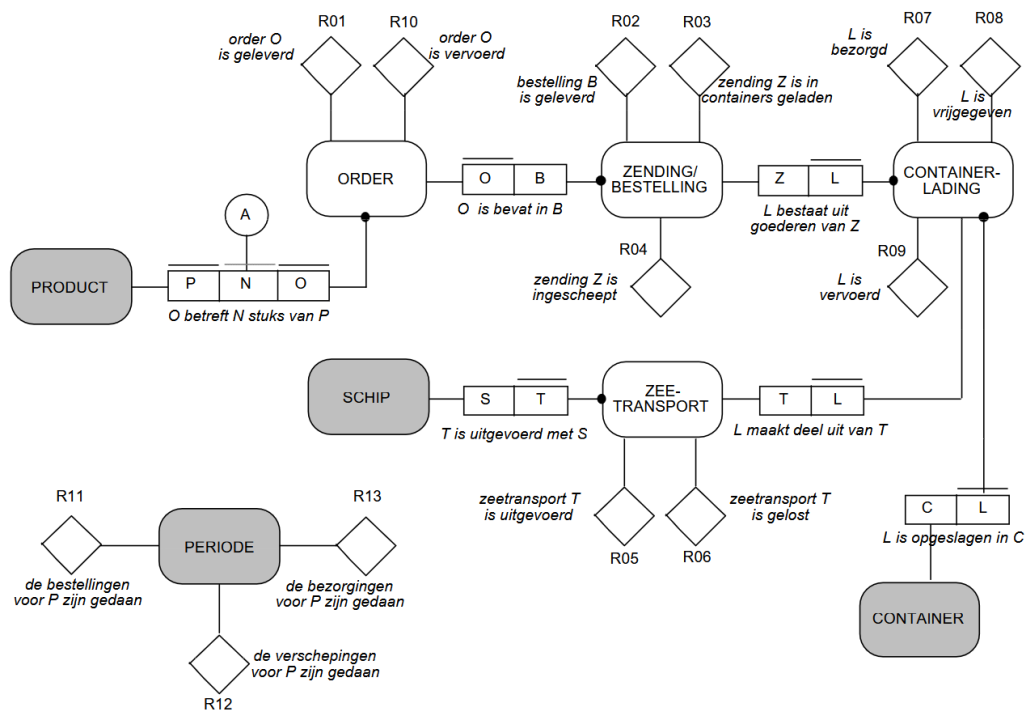


Figure 2.11: DEMO-2 Fact Model

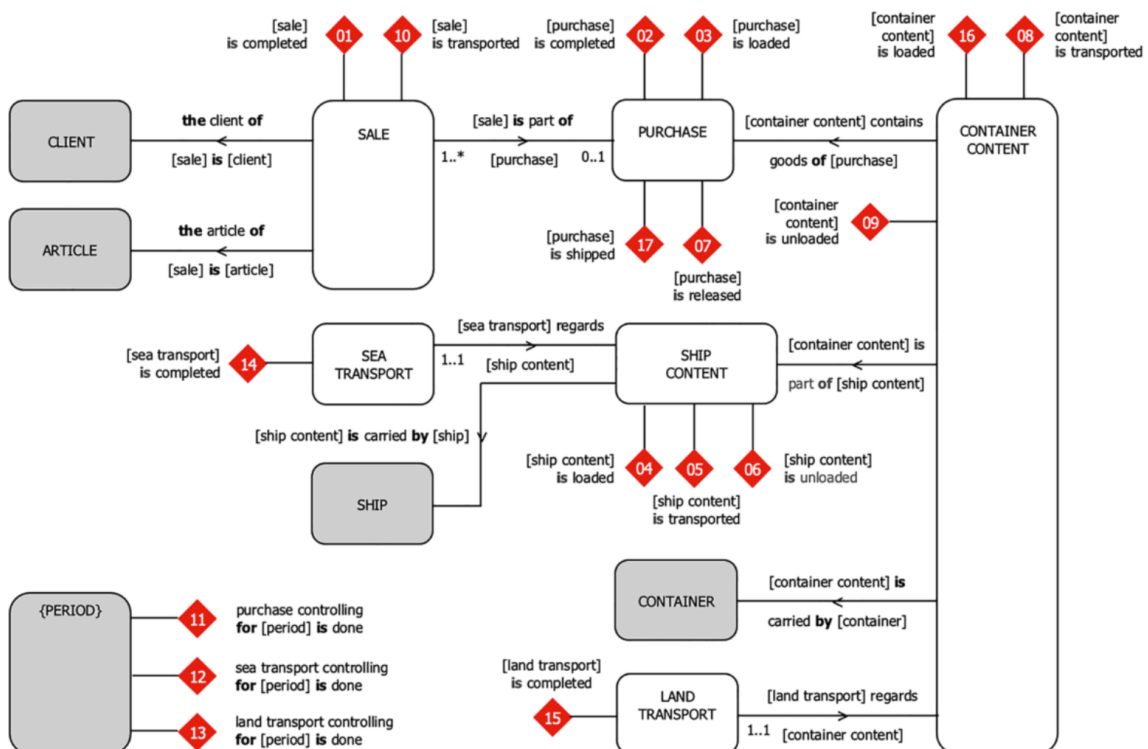


Figure 2.12: DEMO-4 Fact Model

In both DEMO-2 and DEMO-4, the Fact Model does not only give a static information model. There is clearly a dynamic aspect to the Fact Model. The production facts of the different transactions are linked to the objects in the fact model. Take for example the PURCHASE object. There are four

facts linked to it.

- the purchase is **completed**
- the purchase is **loaded**
- the purchase is **shipped**
- the purchase is **released**

If we decide to use the MERODE approach and model the objects as state machines, the production facts are a first indication of the *states* that are required in the state machine.

If we combine this with the full CM model of 2.10, we can even derive more. From the CM we know for example that a certain transaction uses information produced in another transaction, hence the fact produced in the second transaction must exist before the fact from the first transaction can come into existence. This gives an additional indication regarding the lifecycles of the different instances of the objects.

The main visual difference between the DEMO-2 and the DEMO-4 models is the fact that in DEMO-2 the relations were still using the explicit ORM notation. Though I personally think this is a very powerful notation, only few people know/use it. The omission therefore increases the readability of the model. This is however not the reason for showing the two models. The reader who understands Dutch (or the reader that counts objects and uses AI) will notice that the DEMO-4 model has an additional object called the SHIP CONTENT. This necessity object was identified after a discussion of the fact model where the concept of existence dependency was used back in 2008. This indicates that the concept of Existence Dependency can be a powerful tool, even for experienced modelers,

2.3.0.2 Building an EDG for the essential model

Once the fact model has been derived from the CM, it will be transformed into an Existence Dependency graph. As indicated above, this may lead to the introduction of a new object. If a new object needs to be introduced, there will most probably be an impact on the CM and the PM too. The creation of the DEMO models (extended with the MERODE concepts) is an iterative approach.

The initial fact model can be derived from the CM (more or less). The CM is created/discussed with the managers. The derivation of the EDG is typically a task for the domain experts. They have a view on the "structure of reality" (Existence Dependency). As will be seen below, the Existence Dependency often depends on certain design decisions. For example if an order is converted one-to-one into an invoice, the invoice can be considered to be Existence Dependent on the order. If invoice can contain parts of different orders it is not. This may be a fundamental decision that needs to be decided by business management causing yet another iteration in the modelling exercise.

To convert the fact model into an EDG, we start with the customer order. This is represented by the SALE object and the supplier order (the PURCHASE object). Every sale is linked to a client and an article. Clients and articles exist independent of the logistical flow. They are therefore at the top of the existence dependency tree. A sale is *for* a client and, according to the FM above, for a single article. This is a bizarre restriction, even in this case. Assume that the supplier produces multiple variants of the same equipment, then we would have to make a separate order per variant. This requirement will be lifted in the extension of the model.

The SALE is Existence Dependent on the client and the article. If either of these ceases to exist, the sale automatically ceases to exist and it cannot be executed. Of course, there must be a process to deal with a case when a specific article is no longer produced while there are still open orders. This is part of a *discussion* flow in DEMO (unhappy flow). Once we have derived the model for the happy flow, we must analyze all possibilities and make the necessary changes to the model.

Next is the PURCHASE, the order with the supplier. Figure 2.13 shows a first attempt for modeling

the relation between the PURCHASE and the SALE. There must be a SALE before there is a PURCHASE, hence we could think the PURCHASE is Existence Dependent on the SALE. However, multiple SALES can be part of a purchase, which is not possible in this model. Making the SALE EDt on the PURCHASE is not correct either. SALEs exist without being linked to PURCHASEs: multiple sales are grouped in a purchase order on a periodic basis.

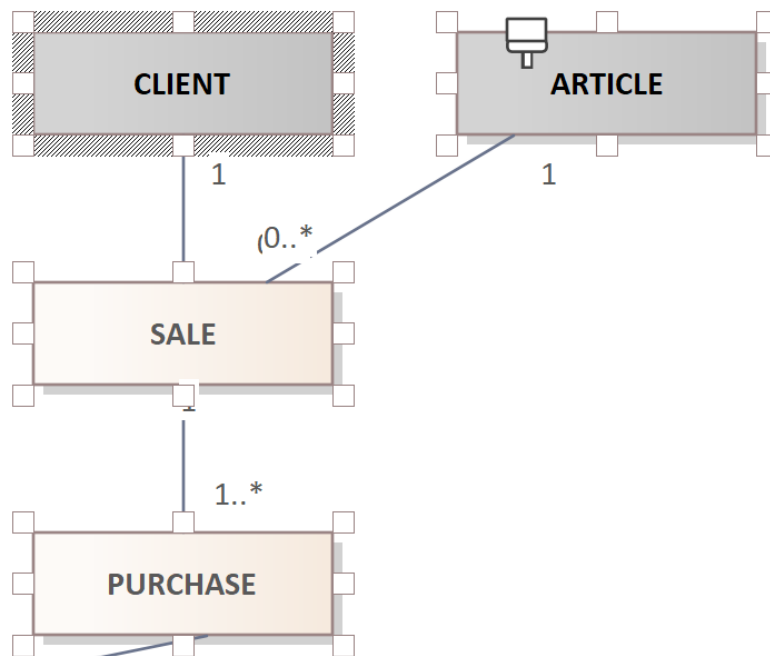


Figure 2.13: Initial EDG

The solution in a MERODE model is to add an object that is Existence Dependent on both the SALE and the PURCHASE (2.14). We called it *Purchase Order for Sale*. This object has its own behavior, which is at least the creation of the link between the SALE and the PURCHASE and the end of the object instance when the final delivery is fully executed.

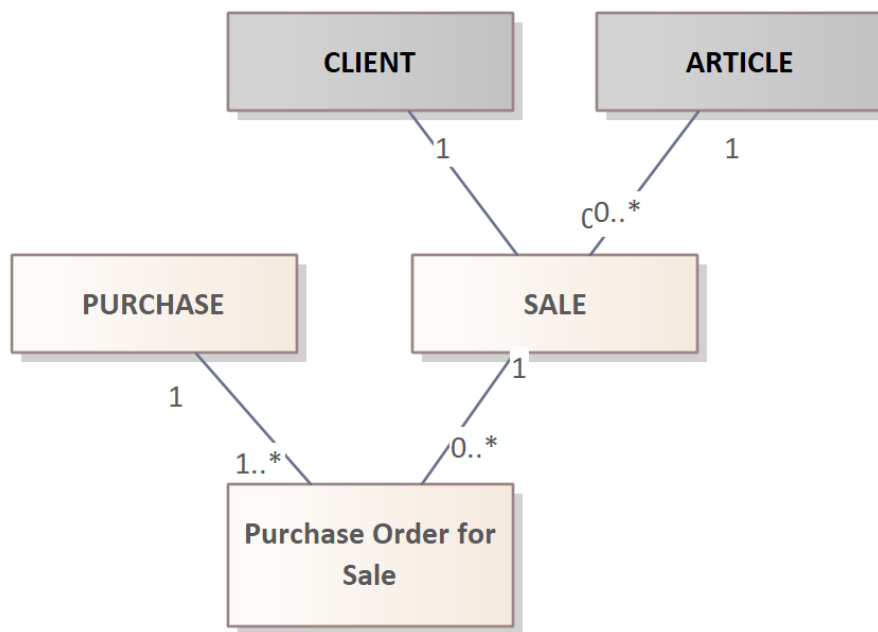


Figure 2.14: EDG with the purchase order for sale object

As already indicated, there is no supplier object in the FM. Still, it is essential to track where the purchase order was placed, hence the supplier will need to be registered as an attribute on the PURCHASE. Following the definition of a business object in MERODE, if we do not create a supplier business object, it is assumed that the supplier does not have any relevant behavior for our system, which translates in DEMO to not having relevant (essential) transactions. Note that we can still have a technical supplier object with addresses, phone numbers and contact information. In DEMO this information would be part of an information bank.

Let us analyze this a bit further: it is reasonable to assume that when you place a purchase order with a supplier, the order must be accepted and you will get a delivery date. This is however something that can be attributed to the PURCHASE and not to the supplier.

The next step is the shipment of the goods. The case states the order is shipped in one or more containers.

As becomes evident in the case description, the (physical) container is an important concept. Every container is handled separately, has its own documents defining what is loaded in the container,... The CONTAINER is clearly an important business object. It has an identifier, its own behavior (created, damaged, destroyed, loaded on ship, unloaded,...).

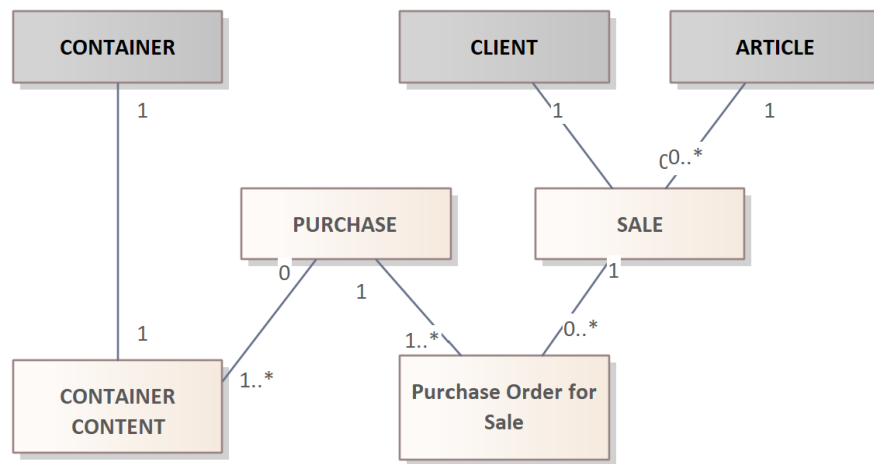


Figure 2.15: EDG with the container object

In the FM, the purchase is linked to the container using the CONTAINER CONTENT object. This object is required because a container is used multiple times. Using Existence Dependency reasoning, it is clear that the life of a container and of a purchase are not directly related. Containers are used for multiple shipments, hence will have multiple contents. On the other hand, the purchase extends over multiple containers and the purchase exists before it is linked to a container. The CONTAINER CONTENT object in the FM thus is Existence Dependent on both the CONTAINER and the PURCHASE. The multiplicity of the relation between the CONTAINER and the CONTAINER CONTENT objects in 2.15. It has been modelled as a 1:1 relation. This means that at any point in time, there can only be goods from a single purchase order in a container. If we allow that the supplier can decide to put goods from multiple purchase orders in a single container, the multiplicity would be 1:1..* instead. It needs to be stressed that the EDG is a view at a specific point in time. As stated above, over time the relation would be 1:0..* (which is not what an EDG expresses).

An additional important consideration about the multiplicity relates to *frozen* and *modifiable* association ends. A frozen association end with multiplicity 1 means that it cannot be changed. This is the standard interpretation in an EDG. The PURCHASE to which a container content is linked can never be changed, the CONTAINER cannot be changed either. If e.g. after loading the container would be damaged and the content is placed in another container, the initial CONTAINER CONTENT instance would be terminated and a new one would be created.

The next step in the process is the shipment. The (physical) ship is definitely a candidate for a business object. In order to do the planning of the land deliveries, IES may want to track the position of the ship. It is not yet clear whether we need to model it as a separate business object or whether it is sufficient to store it as an attribute and have a technical object. I would assume that IES will want to follow up what the ship is doing (docked, on sea, blocked in the Street of Hormuz, ...), hence it seems advisable to choose for the option of a business object.

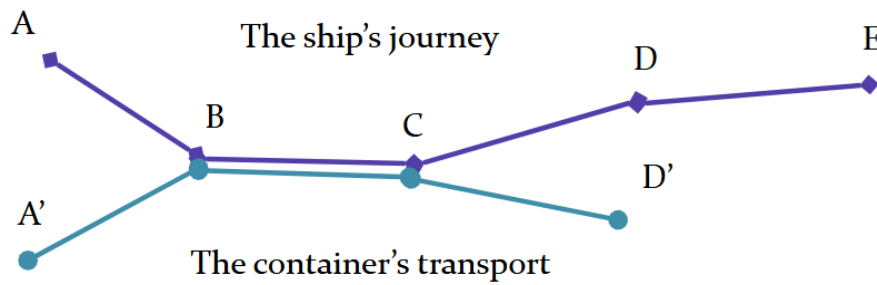


Figure 2.16: The journey of the ship

In figure 2.16 the relation between the container and the ship is shown. A ship makes multiple journeys. Such a journey will bring it at different harbors and in every harbor containers can be loaded and unloaded. Every time the SHIP CONTENT changes. Only when the container is loaded on the ship, there is a relation between the SHIP CONTENT and the CONTAINER CONTENT. It is clear that the CONTAINER CONTENT is not Existence Dependent on the SHIP CONTENT. It is actually the other way round. We are not modeling the operations of the shipment company. We only need to express the fact that our container (with its content) is on the ship that is sailing from port B to C. The SHIPMENT CONTENT thus is the link between the SHIP and the CONTAINER CONTENT and is Existence Dependent on both. Note that you cannot directly link the container content to the ship as no Existence Dependency exists.

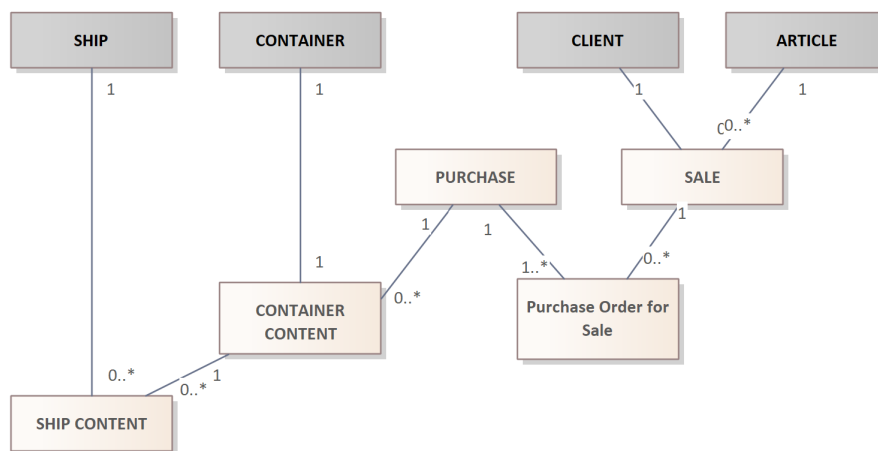


Figure 2.17: The EDG with the ship content object

The two objects left in the FM are the *sea transport* and the *land transport*. These objects have a 1:1 relation with the SHIP CONTENT and the CONTAINER CONTENT respectively in the FM. As these objects have been withheld in the essential model, we should include them in the EDG.

- the SEA TRANSPORT is about the movement of a number of containers, by ship, from a harbor in the Far East to the harbor of Rotterdam.
- the LAND TRANSPORT is about the movement of a single container (with its content) from the harbor of Rotterdam to the warehouse of IES

The sea and land transport are not ordered by IES.

With respect to the land transport, the IES warehouse will be notified that a truck with a container will arrive. IES will have to assign a loading dock at the warehouse and plan a team to unload

the goods, as the case states that the empty container is immediately returned with the truck that transported to container to IES. This means that there is additional information to be registered. The land transport has a different lifecycle than the container content. It is related to the transport of the container content. It is not related to the container. If the container content would be Existence Dependent on the land transport, that would mean that the container content can only exist if there already is land transport, which is clearly not the case. The LAND TRANSPORT object is Existence Dependent on the CONTAINER CONTENT object. The multiplicity chosen is that there is at most a single land transport for the container.

The sea transport is a bit more complex. One could reason that m.m. the SEA TRANSPORT is Existence Dependent on the ship content and not on the individual CONTAINER CONTENT. Because of the relation between the ship and container content, there is an indirect Existence Dependency. Practically this means that e.g. IES will get a message from a shipment company that their containers have been loaded on a ship for shipment to the port of Rotterdam (the Existence Dependency between the two content objects). From that moment on they will follow up at the aggregated SHIP CONTENT level.

In the case it is assumed that the journey will be made by a single ship. So one could argue that the sea transport must be Existence Dependent on the ship.

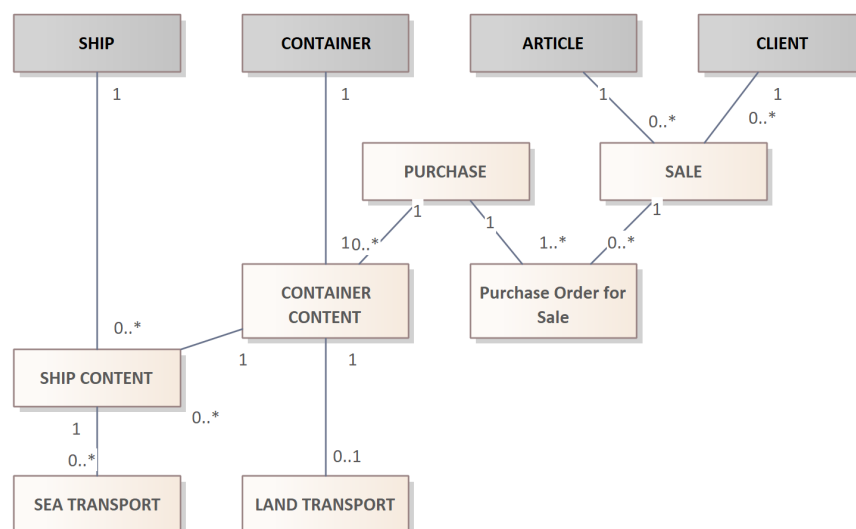


Figure 2.18: EDG for the IES case

2.3.1 Deriving the Object Event Table (OET)

2.3.1.1 What is the Object Event Table?

The concept of the OET has not yet been introduced, while it is an essential deliverable of MERODE. In the OET, the business events are listed as well as the objects on which they have an effect. Here we need introduced some theoretical concepts that are part of the MERODE methodology. Let us take the final EDG for the IES case (Figure 2.18).

- In MERODE, every business event has an owning event. The event to create a purchase order for a specific sales has as owning object obviously the object with the same name. This is called an **owner participation**.
- The MERODE **Propagation Rule** states that every business event can impact all the parent objects of the owning objects, but no other objects than those. These are called **acquired**

participations.

In the example of the creation of the "purchase order for sale" event, this event can also impact the PURCHASE, SALE, CLIENT and ARTICLE objects. For example, when the purchase order is created, some characteristic of the ARTICLE can be updated to show the number of instances of that article have been ordered. As already mentioned previously, to ensure consistency, the implementation of the system must be such that all acquired participations are executed together with the owner participation.

The propagation rule also states that the event will not impact e.g. the CONTAINER object of the SHIP CONTENT, because these are not parent objects. The propagation rules therefore limits the impact of a change to a model. If an additional event is required, the impact will not ripple through the entire model. If a new object is needed, multiple new events will be needed. The impact will however be constraint to a specific part of the EDG. The reader probably understands that this is much in line with the "ripple effect" described in Normalized System Theory.

Some people may object to this approach thinking that it is not possible to build software when one had to adhere to this restriction. Fortunately, about 30 years of experience with the MERODE approach has proven that one can build real, complex system when adhering to the propagation rule.

An example of an OET for the library example from [5] p.116 is the following:

	MEMBER	TITLE	COPY	LOAN
register_member	O/C			
stop_membership	O/E			
cr_title		O/C		
end_title		O/E		
acquire		A/M	O/C	
classify		A/M	O/M	
borrow	A/M	A/M	A/M	O/C
renew	A/M	A/M	A/M	O/M
return	A/M	A/M	A/M	O/E
sell		A/M	O/E	
lose	A/M	A/M	A/E	O/E

Figure 2.19: Owner and acquired participations for the library example

As a business object in MERODE must have at least a create event and an end event, for every object (column in the table) there must be at least an even tagged with a C and one with E. d

2.3.2 Deriving the events and the OET

The fact model presented above (Figure 2.8) contains a link between the object and the transactions that produce facts related to these objects. This is summarized in the table below. Four columns are greyed out: SHIP, CONTAINER, ARTICLE and CLIENT. The CM does not contain a transaction that impacts these objects. These are the objects that exist external to IES. You can debate whether you want to include them as a business object in the MERODE model or not.

We have opted to include the four objects in the model as business objects. We must define at least a create and end event. This will allow us to extend the behavior (add transactions) in a later phase.

For example, one could assume that a client needs to undergo a regular credit check process, clients could be blocked for future orders, ... The container object seems of a different nature. We need of course the container number and we will get documents related to that container. These are in reality all connected to the container content. As IES we do not really care about the container. Hence, it may make sense not to create this business object.

			Article	Client	Sale	Purchase	Purchase order for sale	Container	Container content	Land Transport	Ship	Ship content	Sea transport
T01	sale completing	[sale] is completed			x								
T10	sale transporting	[sale] is transported			x								
T11	purchase controlling	purchase controlling for [period]											
T02	purchase completing	[purchase] is completed				x							
T03	purchase loading	[purchase] is loaded				x							
T17	purchase shipping	[purchase] is shipped				x							
T07	purchase releasing	[purchase] is released				x							
T12	sea transport controlling	sea transport controlling for [period] is done											x
T14	sea transport completing	[sea transport] is completed											x
T04	ship content loading	[ship content] is loaded										x	
T05	ship content transporting	[ship content] is transported										x	
T06	ship content unloading	[ship content] is unloaded										x	
T13	land transport controlling	land transport controlling for [period] is done								x			
T15	land transport completing	[land transport] is completed								x			
T16	container content loading	[container content] is loaded							x				
T08	container content transporting	[container content] is transported							x				
T09	container content unloading	[container content] is unloaded							x				

Figure 2.20: Events derived from the fact model

The life cycle of a business object is of course not only defined by the production facts. The different process models give a more detailed view. We focus on the core process (T01 and T10) in this discussion. Figure 2.21 shows the process flow in the DEMO language. This is the "happy flow". As explained in the previous chapter, there is also a (structured) discussion flow.

According to the Enterprise Engineering theory, underlying the DEMO methodology, the happy flow follows a standard communication pattern:

- the client makes a *request* to have an article delivered
- IES *promises* the delivery
- IES orders the goods in the Far East and has them delivered to the client, this is red diamond 01 in the diagram (the production fact)
- IES *states* that the delivery is made (the production has been realized)
- the client accepts the delivery (implicitly or explicitly)

When mapping to a MERODE model, it is assumed that the initial request (T01/rq) creates the SALE object. Even if the SALE would not materialize because of a *discussion* (e.g. the article cannot be delivered), we need to track the intention of the sale. When IES commits to delivering (T01/pm), the state of the object changes. The promise event triggers a request to T10. This transaction is concerned with the delivery of the goods to the client. The process model shows that the transaction starts when the promise of T01 occurs. Another option would be to wait for the delivery (shipment) of the goods (T02). In that case however, the planning of the unloading and transport could only start later. It seems the better option to start this transaction early in the process and e.g. log the estimated delivery (arrival) on this object for planning purposes.

In a MERODE context, the events T01/pm and T10/rq are the same. No additional states on the SALE object are required.

The next event on SALES is the production fact of T10, which is the delivery of the goods to the client. In the diagram you note an arrow T02/ac, a so-called "interimpedement" link. This type of links link the different processes. It means that the production of T10 is realized when T02 is accepted. T02 is the purchase and delivery of the goods. It can be assumed that unloading the containers and controlling the goods is part of the responsibility T02. This implies that repacking the goods and transporting them to the client is part of the responsibility of T10. It is important to notice that the model does not contain any information about all the actions required to realize the delivery, DEMO is not concerned with the details of the production but with the coordination of the events. This leaves us with a couple of choices with respect to the refinement of the requirements:

1. We extend the essential model
2. we model this directly in MERODE

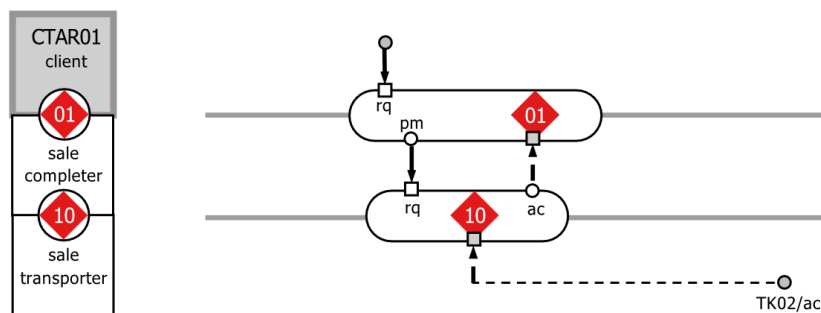


Figure 2.21: Process model of the sale

Like in all real cases, there are many parameters to consider when making this choice. My personal preference is though that, if management decisions are still needed, I make a more detailed cooperation model, with additional commitments and responsibilities. That is based on my experience that can discuss a DEMO-like drawing or description with business people. If that is not the case I prefer to create the model directly in the MERODE language. This is less work and - for an IT nerd - more accurate. In the IES case I would assume that a link with a WMS (Warehouse Management System) needs to be made, which is of technical nature. Therefore, I would not consider additional DEMO transactions.

T10 ends with the acceptance of the delivery of the goods by the client. There should be a document signed by the client (or an electronic message) that confirms that the delivery was made. The registration of this acceptance T10/ac triggers the production of T01: the completion of the sale, followed by T01/st stating that the sale is completed. One can assume that this event triggers the creation and delivery of an invoice. In that case, the payment of the invoice could be considered as the acceptance T01/ac that the sale is finished. There is another possibility: T01/st could be a message stating that everything has been delivered, followed by an implicit acceptance. As stated above, IES is an internal service of a multinational company and is not concerned with the invoicing and payment. This point at the second possibility with the implicit acceptance

Figure 2.22 shows the state machine of the SALE object, documenting the happy flow. Obviously, additional transactions need to be added. For example, an order can be modified as long as it is not accepted by IES. This is part of the discussion flow. It is easier to add the state transitions and in the OET than to draw the DEMO model. It is however advisable to use the discussion template from DEMO as a guideline.

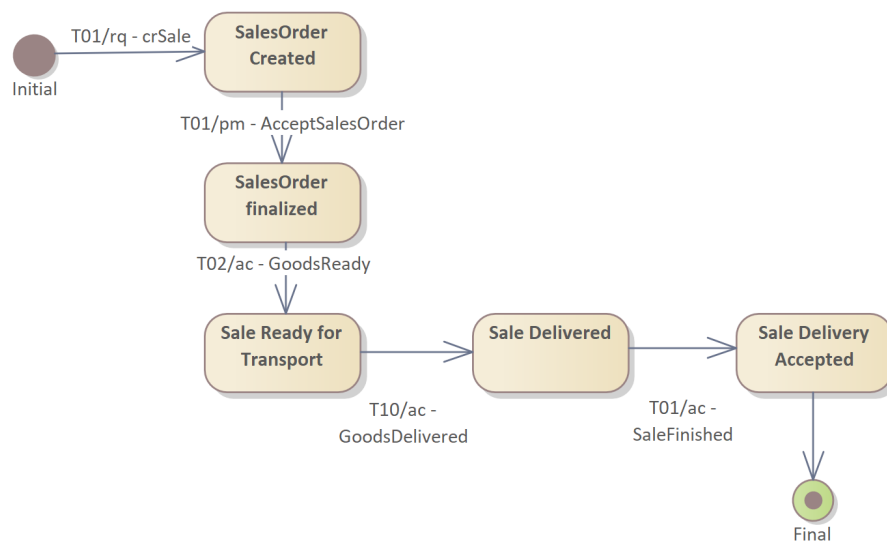


Figure 2.22: Happy flow State Machine for the Sale Object

A last important point is about ending the SALE object. Ending an object does not automatically mean it is deleted from the system (on the contrary). It means that the lifecycle has ended and that it cannot be changed anymore. It should therefore be considered not to use the event T01/ac as the end event, especially not when it is an implicit acceptance. It may be possible that things happen that impact the sale. For example, assume that the supplier delivered the correct boxes but put the wrong equipment in the box. Even if the detail is registered on a different COMPLAINT object that is added to the model, you may want to impact the SALE object as well for later analyzes purposes. In case of GDPR sensitive data, you may have to abide to strict rules for deleting the data after a specified period of time. You could decide to use the end event to indicate that an object instance is ready for deletion.

2.3.3 Payment

The IES case does not include a payment. As explained above, IES is an internal service of a multinational group and is not concerned with the payments. If a payment is required, a payment transaction needs to be added to the essential model. A management decision regarding the timing of the payment is required, as there are different possibilities:

1. orders need to be paid in full by the client before the goods are ordered from the supplier
2. goods are paid after delivery
3. new customers need to pay in advance, returning customers with a good credit history can pay after delivery
4. ...

Adding a payment impacts all DEMO submodels: a transaction is added in the CM, potentially a fact (the payment object) is added in the FM. The choice regarding the timing of the payment will have an impact in the process model (PM). Note that to model the assumption of 3, two different payment transactions will need to be added, each linked to a different type of customer. In the action model (AM) the relevant business rules can be added.

In DEMO, the transaction related to the payment creates the fact "[order] has been paid". It does not refer to a payment object. This is in line with MERODE: an incoming payment is an event. It does not have behavior that can be modeled with states and events causing state transitions of the payment. Unless if you are a bank, an incoming payment is a line on a bank statement that impacts

the state of another object like the invoice that has been paid.

An outbound payment is something completely different, e.g. payments may need to go through an approval process, next they may end up on a list to be loaded in the software of the bank, they reach their final state when the bank confirms that the payment has been made. It may also happen that the payment is not executed because of insufficient funds, passing a daily payment limit or because of a wrong target account.

If we use the assumption that the order needs to be paid in full, a payment transaction (T20) needs to be added. The sale completer will request the client for payment. The related production fact is [SALE] has been paid.

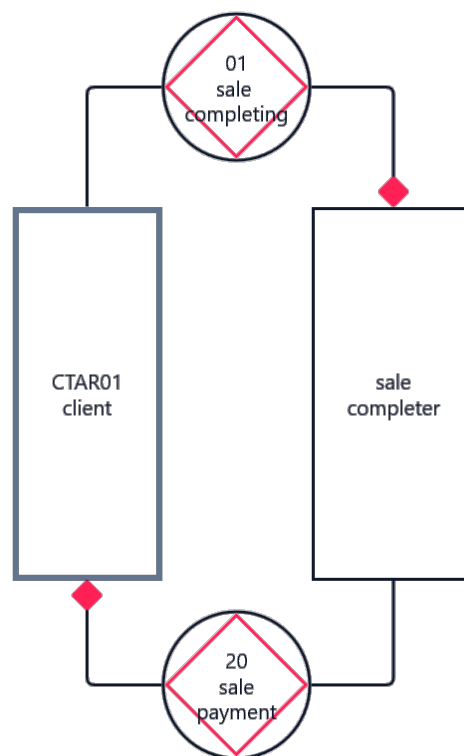


Figure 2.23: CM with payment transaction

In the PM we will need to show that the sale completer will only promise to deliver the sale when the payment has been received, i.e. when the production fact of T20 has been produced. The PM is shown below. This should be compared to the original process model of the sale in figure 2.21. The "promise" event now consists of two sequential actions. First the payment is requested. If the payment is accepted, the request of transaction 10 is triggered.

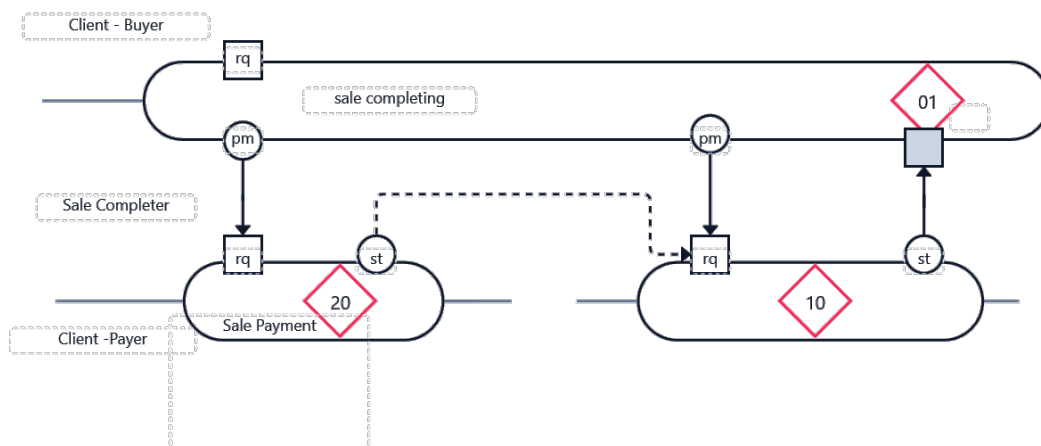


Figure 2.24: CM with payment transaction

When we translate this into a MERODE model, we will also need an additional state in the SALE object, linked to the establishment of the production fact of T20. Only when the object has reached this state, the process can continue. If we decide to model the payment with a business object, we will need to add a PAYMENT object in the EDG. If the client has to be pay per individual sales order (like in a webshop), the payment can be Existence Dependent on the SALE. If the client can pay multiple orders at the same time, a "PAYMENT FOR SALE" object needs to be introduced.

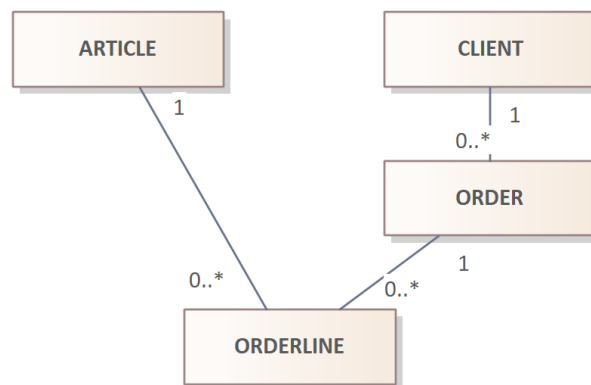
2.3.3.1 The client order

The FM of the IES case has the following relation: "ARTICLE is the article of SALE". This indicates that a sales order is related to a single article. In the FM, this restriction can easily be eased by adding a 1..* multiplicity at the ARTICLE end point and renaming the relation, pretty much like the SALE-PURCHASE relation (1 purchase can be related to multiple SALES).

In an EDG, it is a bit more complex. The concept of the order line linked to the order has to be introduced.

- The order is linked to a client (only one). Clients can of course exist without having made orders. An order cannot be relinked to another client.
- An order can be created without order lines. This will allow us to enter information about the customer's expectations before it has been decided which articles will be ordered
- Assume that we want to store the stock levels on the articles - assume e.g. that IES has an allotment of every article that can be ordered. This means that every time an article is ordered by the client, the stock level needs to be updated
- An order line is Existence Dependent on a single article, an order line is linked to a single order because it is Existence Dependent on the order

This is expressed in figure 2.3.3.1.

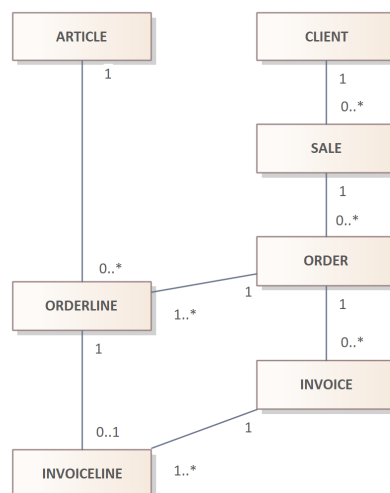


If you allow multiple articles on an order, you may need to think about organizing the deliveries on a line-by-line basis so that the available goods can be shipped once available, while the unavailable goods can be shipped later. This adds the requirement to record which order lines are ready for shipment to the client and which are still on the to-be-shipped list. Orders that have been fully shipped will also be marked so that you know that the order does not need further processing. The evolution of the MERODE model is handled in detail in the MERODE course notes of Prof.Snoeck.

2.3.4 The invoice

The IES case does not mention an invoicing process. IES is assumed to be an internal service responsible for the shipment of the goods and not for the financial flows. While not mentioned in the case, it can be assumed that some other part of the organization bears this responsibility. Pretty much like the ordering process, the invoicing process can be considered to be part of the sale transaction. The invoice creation can be linked to transaction step T01/st when IES "states" to the client that the goods have been delivered. Let us stick to the assumption that the payment is required before the order is treated by IES. That still leaves us a lot of possible implementation that would all be compliant with the essential (DEMO) model.

A first possibility is shown in the next diagram in which the link with the SALE object has been added.



- An order is for a specific SALE, linked to a specific customer.
- When the order is created, it is required to add at least one orderline. Any information to be logged before there is an actual order, will be logged on the SALE object.
- The invoiceline is linked to the orderline with the multiplicity 1/0..1. This means that an orderline is invoiced or not. In case of a partial delivery of the orderline it cannot be invoiced yet as we cannot have multiple invoicelines for a single orderline. A business rule stating that only fully delivered orderlines can be invoiced must be added.
- The invoice is linked to the order. It is specified that there can be multiple invoices for a single order. This means that an invoice can be made even if an order is only partially delivered. If we would require that only a single invoice is created per order, the multiplicity of the relation would be 1:1 and another business rule would need to be added. In that case one could argue that there is no need for a separate invoice and invoiceline object.
- Because there are two paths from invoiceline to order (one via orderline and one via invoice), a rule must be added that they must link to the same order.

In this model it is not possible to invoice for goods that were part of different orders. If that is what we want, e.g. to support periodic invoicing based on everything delivered, the MERODE model changes. The invoice is not Existence Dependent on the order anymore. But, the invoice will not be Existence Dependent on the sale either, because an invoice can contain lines belonging to different sales. This implies that the invoice object cannot be contained in the sale object. In fact, a fifth process would need to be added in the CM. We would need to add a self-initiating transaction for the periodic invoicing, with a responsible role for the creation of the invoices. The invoice object will need to be added to the FM of the essential model.

2.4 Conclusion

I hope that this section has demonstrated that DEMO and MERODE fit together and gives an impression of the way of working and how things fit together. The detailed derivation of a complete MERODE model for an ordering and invoicing flow can be found in the course notes from Prof. Snoeck. The reader will notice that these model can easily be reused in the IES example. Existence Dependency models are "universal" - it is highly unlikely that A is existence dependent on B in a certain business domain while it is the opposite in another. The business implications of A being Existence Dependent on B are similar in all domains.

The strength of the integrated methodology is that it starts from the four DEMO models. By converting the Fact Model on an EDG, the model is refined. This is done together with the domain experts who have a good understanding of the universe of discourse. Because the concept of Existence Dependency is easy to understand, communication with business people is much easier than when trying to discuss an UML model with its variety of relationships. Just try to explain the difference between the aggregation and composition relation.

By integrating the two approaches we can start from a high level model describing commitments and responsibilities and working our way down to the level of objects and events *in a consistent way*. The resulting models can be fed into the Normalized Systems expanders to generate evolvable software.

Of course, one could model the system directly into Normalized Systems to create the software. In that case we would miss the bridge with business model. The quality of the output would solely depend on the quality of the analysts that came up with the model. The biggest advantage of the DEMO-MERODE approach is however, in my opinion, the creation of a stable information model. Changes to this information model because of changing requirements do not ripple through the entire model, but would have a localized (bounded) impact.

Bibliography

Books

- [1] Jan De Winter. *Concise Guide to Enterprise Transformation*. Springer Nature, 2026 (cited on pages 4, 8, 9, 24).
- [2] Jan LG Dietz and Hans BF Mulder. *Enterprise ontology: A human-centric approach to understanding the essence of organisation*. Springer Nature, 2024 (cited on pages 8, 24, 35, 37).
- [3] Jan LG Dietz and Hans BF Mulder. *Enterprise design fundamentals: Settling an enterprise business and devising the enterprise organisation*. Springer Nature, 2025 (cited on page 8).
- [4] Jan LG Dietz and Hans BF Mulder. *Enterprise Governance and Management: Setting a Route for an Enterprise and Keeping It on the Route*. Springer Nature, 2026 (cited on page 8).
- [5] Monique Snoeck. *Enterprise information systems engineering*. Springer, 2014 (cited on pages 11, 45).

Articles

- [6] Monique Snoeck and Cindy Michiels. “Domain Modelling and the Co-Design of Business Rules in the Telecommunication Business Area”. In: *Katholieke Universiteit Leuven, Open Access publications from Katholieke Universiteit Leuven* 4 (2002). DOI: [10.1023/A:1019914823229](https://doi.org/10.1023/A:1019914823229) (cited on page 11).
- [7] Dirk Van Der Linden et al. “Towards evolvable state machines and their applications”. In: *Int. J. Adv. Syst. Meas* 6.3 (2013) (cited on page 14).