International Journal of Academic Scientific Research
ISSN: 2272-6446 Volume 3, Issue 4 (November-December 2015), PP 76-100
www.ijasrjournal.org

Enterprise Ontology-based Information Systems Development

Tarek Fatyani [1], Junich Iijima[2], Jaehyun Park[3]

1, 2, 3 (Department of Industrial Engineering and Management, Tokyo Institute of Technology, Japan)

***Abstract**: Information Systems Development (ISD) has encountered a variety of challenges in identifying complex requirements from multiple stakeholders. Therefore, users and their information environments have been a central issue for developing an abstract model of the early stages of information system development. To address this issue, modeling methodologies have considered modeling the enterprise as a social system. DEMO is a social system modeling methodology that addresses diverse social theories and multiple stakeholders' requirements. Additionally, the DEMO model has improved not only the quality of the requirements but also reduced the time for implementation during the early stages of ISD. Therefore, this research focuses on developing an information system based on the DEMO model by deriving the conceptual artifacts of an IS from the DEMO model. Based on numerous real world projects, this study proposes that use of the DEMO model could lead to important discussions associated with actor roles, responsibility, and authority. These discussions are critical when defining the objectives and requirements of any information system to be developed.*

***Keywords** – DEMO Model, Information Systems Development, Requirements Engineering.*

I. Introduction

As enterprises have grown in size and complexity, the complexity and pervasiveness of their Information Systems (ISs) have grown exponentially [1]. In addition, continual advances in technology require continuous improvement in existing ISs. Due to this complexity, IS project failures are an ongoing issue because of the misalignment of business and IT. For example, the Standish Group reported that only 32% of IT projects are successes [2], while Wright and Capps [3] stated, "20 – 30% of all IS development projects are perceived as overwhelming failures, while 30% to 60% are partial failures." One cause of business/IT misalignment is that IT systems' functionality often does not meet expectations. For example, a research case study of a UK public sector organization found that an estimated 60% of completed projects had not met the original objectives [2]. Another cause of business/IT misalignment is the need for major modifications to IT systems after acceptance testing, which may result from the lack of early validation for IS functional requirements [2]. Furthermore, IS complexity may result in cost overruns that exceed estimates [4]. Additionally, the cost of enterprise IT system maintenance increases exponentially over time, sometimes becoming unmanageable in terms of both complexity and resources [3].

Therefore, developing ISs successfully is extremely challenging due to their complexity. It is generally agreed that the only realistic way to manage this complexity and continue to provide IS benefits is to develop ISs using appropriate methods of abstraction [4]. This goal requires an abstract model that enables people to understand an enterprise in a simple and concise way. Abstract models are used primarily as communication and

collaboration mechanisms when groups need to solve problems and/or obtain joint and mutual understanding of an overall design [5]. The use of abstract models for IS design promises improved productivity, product quality, and shorter lead times because of increased abstraction levels and reduced gaps between problems and solutions. Abstract models are also useful for maintenance and technical software documentation [6, 7, 8, and 9]. These gains could be achieved by modeling the enterprise at a stage prior to the early development stages of an IS.

Although many modeling methodologies currently exist, including UML, IDEF, BPMN and EPC, current modeling methodologies are not sufficient to meet the IS community's needs [10]. UML's complexity is another recurrent theme, especially the complexity attendant upon large models, a criticism that echoes through the literature. However, despite their maturity, these modeling methodologies still lack comprehensive constructs for representing some core business concepts [11]. Although such modeling methodologies are supposed to support understanding the business domain using a more abstract model that is completely independent of implementation, these methodologies consider both the original facts and the implementation decisions made during the design phases of IS development, but do not differentiate between them, which increases the complexity of IS development efforts. A relatively recent approach to coping with the aforementioned problems was proposed by Pandey et al. [12]. They developed i* (i-Star), a modeling methodology that can represent how well the intended system meets organizational goals, why the system is needed, what alternatives were considered, what the implications of the alternatives were for the various stakeholders, and how the stakeholders' interests and concerns might be addressed. Very recently, in [13], a comparison between i* and Design & Engineering Methodology for Organizations (DEMO) showed that both are social modeling methodologies: they focus on humans and human interactions in their modeling. DEMO is implementation independent; therefore, the DEMO model does not need to change before or after implementing any IT solutions. However, i* is an implementation dependent methodology. DEMO provides a more formal and rigorous model of the enterprise, which makes it a good potential modeling methodology for understanding an enterprise before implementing any IT solution.

DEMO is built on the Performance in Social Interaction (PSI) theory and provides a consistent, coherent, concise, comprehensive and essential (C4E) representation of the system. The DEMO model is both abstract and implementation independent, and it tends to provide a mutual understanding of the business. DEMO has already proved its usefulness in enterprise engineering, yet developing ISs based on DEMO is still challenging. These problems occur precisely because DEMO is an implementation independent model. DEMO does not consider methods of implementation; instead, it focuses on the ontological level of the enterprise. Therefore, implementation-related design decisions are deferred and must be made during IS development. There are several studies related to using DEMO for IS development available in the literature, but these studies discussed only the theoretical side and provided no guidelines or practical advice on developing IS-suitable DEMO models or transferring existing models to an IS.

This paper provides a practical framework to illustrate the stages involved in using the DEMO model and for developing ISs based on it. Furthermore, this paper provides practical guidelines for developing DEMO models that function as good abstract models for developing ISs. The remaining sections of the paper are organized in the following way: Section 2 provides a review of the current literature concerning process modeling

approaches that focus on DEMO. Additionally, it provides a critical review of the most closely related work. A brief overview of DEMO, the proposed framework, and some guidelines are presented in Section 3. Then, a real world case study is introduced in Section 4. Related discussions, results and insights are presented in Section 5. Finally, conclusions and future work are covered in Section 6.

II. Literature Review

Enterprise ontology is a core theory of enterprise engineering. In particular, the goal is to provide a new understanding for enterprises, penetrating the distracting and confusing appearance of an enterprise and revealing its underlying essence. Enterprise ontology DEMO models are based on the PSI (Performance in Social Interaction) theory. In PSI theory, an enterprise (organization) is considered to be an interaction of individual social subjects [4]. DEMO helps in "discovering" an enterprise's ontological model primarily by re-engineering it from its implementation.

In DEMO, an organization is composed of actors (human beings) who perform two types of acts: production acts (P-acts), through which the actors contribute to bringing about the goods or services that are delivered to the environment, and coordination acts (C-acts), in which actors enter into and comply with commitments, initiating and coordinating the execution of production acts. C-acts and P-acts occur in universal patterns called transactions. A transaction involves two subjects: an initiator, who generates a request, asks for a production and commits to the delivered result, and an executor, who produces products, gives feedback to the request and is responsible for producing the result [4]. Transactions are the elementary (essential) organizational building blocks of enterprises. There are three patterns of transactions: basic, standard and complete. The basic pattern includes only the happy path, while the standard pattern covers negative acts such as "refuse" and "decline." The complete pattern includes both completion and revocation acts, allowing actor roles to revoke their acts. DEMO differentiates between three transaction levels: ontological, infological and datalogical. An ontological transaction is one that creates the original artifact as a result of executing it. An infological transaction involves a derived artifact such as a calculation. A datalogical transaction is a transaction that merely involves storing and retrieving data without changing it. In DEMO, ontological transactions are denoted by the color red, infological transactions by green, and datalogical transactions by blue.

Developing ISs is a challenging task because of their complexity. It is generally agreed that the only realistic way to manage this complexity and to continue to provide IS benefits is to develop ISs using appropriate methods of abstraction [14]. The literature is rich with a variety of modeling methodologies that support the many different modeling needs in IS development. First, we present and then briefly discuss the literature. Second, we provide a critical review of the papers most closely related to our work.

Business process modeling is essential in IS development. During the early stages of the IS development cycle, a domain model is a type of artifact that captures the common and variable aspects of software products [15]. A study conducted by Shishkov & Dietz, 2001 [16] showed that use cases represent a promising and essential tool for business process modeling because they can help with visualizing models of the business processes under study. The study also proposed that use cases could be combined with activity diagrams for building more consistent and complete models of a system and representing different system-actor interactions. However, use

cases are intended to capture the functional requirements of ISs; identifying use cases at the model stage is not so simple. To tackle this issue, Dietz, 2003 [17] proposed an approach that derives use cases from business system models produced by applying Demo Engineering Methodology for Organizations (DEMO). To derive the use cases from the models, they proposed a three-step procedure by which the derived use cases retain the same properties of essence, atomicity and completeness found in the models. Furthermore, recent research by Bera & Evermann, 2014 [18] specified the semantics of various UML language elements for domain modeling, based on ontological considerations. They empirically examined ontological modeling guidelines for the UML association construct, which plays a central role in UML class diagrams. Using an experimental study, they found that some of the proposed guidelines led to better application domain models. They also found that ontological guidelines could improve the usefulness of UML class diagrams for describing the application domain, and thus have the potential to improve downstream system development activities and ultimately affect the success of an IS implementation.

A relatively recent approach, i*, has been applied by many authors to cope with the aforementioned IS development problems. Alencar et al. [19] used the i* models to identify crosscutting concerns early in the software development process. To accomplish this, they started by analyzing the strategic dependencies and strategic rational models to identify those model elements that might cut across several other elements. The result was validated by applying a set of heuristics to derive a use case model from the i* models and then identifying potential crosscutting concerns in the resulting UML model. For business/IT alignment, Maz´on et al. [20] used the i* modeling framework and the Model Driven Architecture (MDA) approach to describe (i) how to model goals and information requirements for data warehouses, and (ii) how to derive a conceptual multidimensional model that provides the information required to support the decision making process. Furthermore, António et al. [21] aimed to benefit Software Product Lines (SPLs) from the i* framework, taking a more expressive approach toward requirements engineering for SPLs. Thus, they extended the i* framework by adding cardinality to the intentional elements, making it easier to derive a feature model from the i* models.

Many authors have applied DEMO to support IS development. One frequent cause of software project failure is the mismatch between the (business) requirements and the actual functionality of the delivered (software) application. Shishkov & Dietz, 2004 [22] proposed an approach to software design that is consistently based on prior business process modeling. The alignment between these two tasks is realized in a component-based manner, by deriving the software model from identified (generic) business components, thus taking advantage of the benefits of object-orientation. However, their study was both theoretical and abstract rather than practical; they expected that the proposed approach would be a useful contribution to the knowledge on aligning business process modeling and software design. In another study [23], authors aimed to discover how to find all relevant use cases, based on sound business process modeling. They studied and analyzed DEMO's strengths concerning the derivation of use cases. Shishkov & Dietz, 2005 [24] extended the SDBC approach by creating step-by-step methodological application guidelines on how to accomplish this. However, their study is also theoretical and abstract rather than practical.

Starting from the notion that the direct mapping of business processes to business components often leads to

inadequate results, a methodology for identifying and refining business components based on the functional decomposition of an application domain was proposed by Albani et al. (2003) [25]. To support collaboration between business partners, adequate ISs must be built to automate inter-organizational business processes. To guarantee the appropriateness and quality of the underlying business domain models, Albani et al. (2006) [26] introduced a process for identifying business components based on enterprise ontology, assuming a business domain model that satisfied well-defined quality criteria. However, their study is also theoretical and abstract.

To avoid a mismatch between the provider's intent and the consumer's expectations concerning the functionality of a corresponding service, Terlouw and Albani (2010) [27] proposed an enterprise ontology-based approach for service definition, service classification, and service specification. They concluded by deriving the information required for each service from the transactions.

To respond effectively to environmental changes, such as changes in market needs, technology, regulations or law, enterprises need to be able to modify their supporting information system(s) accordingly. Krouwel and Land (2011) [28] aimed to find key concepts to link agile enterprises with agile automated ISs by combining DEMO and Normalized Systems. They found that DEMO and its underlying PSI-theory match the principles and elements of the Normalized Systems approach. Also, using two cases of Dutch governmental subsidy schemes, they designed a few automatable steps to derive a Normalized System from the ontological model of the B-organization, provided by applying DEMO to an enterprise, while retaining the implementation freedom of the organization under consideration. However, their approach is too complex, and there are no guidelines.

In the available literature, three publications are closely related to this study. We discuss these critically below.

Albani and Dietz (2011) [29] demonstrated an Information System Development (ISD) methodology based on the notions of enterprise ontology and business components, and explained it within a conceptual framework called the generic system development process. The methodology allowed both for reduction in the complexity of domain models and for identification of stable business components. Furthermore, the resulting IS models contain only their essential features, which appear to be quite understandable by business people. However, there are some shortages in their methodology. First, they presented only general concepts about function and construction perspectives. Second, it is too abstract; concrete details about actual performance are lacking. Third, the relation between green and red is not simple; instead, it is a network.

Guerreiro et al. (2013) [30] conceptualized an architectural framework specifically designed for explaining and representing the working environment for enterprise operating systems (EOS) and non-EOS ISs that depend on them. They proposed a new conceptual structure for software architectural frameworks. Such frameworks are intended to reduce complexity and ambiguity during software engineering for several classes of IS. Their proposed framework is a good simulation tool, is easy to model, and its DEMO processor is easy to implement; however, there are some omissions. First, the framework did not considered the rational aspects of ISs. Second, it focused only on the CM and the PM, and did not use the FM. Third, it captured only the coordination facts—not the production facts. Finally, it must be linked with other systems.

Jong (2011) [31] proposed the PID framework, which guides the design of enterprise information systems based on enterprise ontology. He provided general guidelines for implementations of PID and also proposed integrating

the three layers of DEMO models—the business, information and data layers. While the relationships between the three layers are interesting, from a practical point of view only the business layer (with a few information layer transactions) should be modeled. Including the entire set of possible transactions and including the data layer only makes the model more complex. Moreover, the original problem involves making the model simpler and focusing on only the important aspects of the enterprise. Another point missing from this research is the linkage between the DEMO models and other IS-related models. This linkage make it very difficult for IS engineers to adopt the system utilizing only the DEMO models.

The following three points sum up the current gaps in the literature related to developing ISs based on DEMO. First, an IS is a mixture of rational systems executed by computers and social systems composed of people interacting with the system. There is a need to capture both the social and rational aspects in the model. Second, most of the research focuses on the theoretical aspects, creating a gap between research and industry. There is a need to justify these theoretical frameworks through real-life cases studies. Third, there is a need to utilize all the aspects of DEMO models in developing an IS, including not only the functional aspects of the IS but also how to develop the database schema from the Fact Model in DEMO.

To fill these gaps, this research proposes a practical framework for developing ISs based on DEMO. A real world case study follows that shows how to utilize the framework.

III. Framework

This section introduces our framework for developing ISs based on DEMO models. The purpose of this framework is to reduce the complexity of the proposed IS during the first stages of the analysis and design phases. This reduction of complexity is not only important for developing good software but also has a crucial impact on the future maintenance of the developed software [32]. Our framework is based on DEMO, which has proved its usefulness in reducing the complexity of the enterprise in both process reengineering and in software development [33, 34, and 35]. The framework derives IS artifacts from DEMO models. Those artifacts are composed of UML because UML is already familiar to IS developers, and the use of UML diagrams has a good impact on source-code comprehensibility and modifiability [36]. Next, we provide workflow guidelines to illustrate a practical method of implementation. We assume that the DEMO model has already been completed in a previous stage. Therefore, this framework does not deal with constructing the DEMO model; instead, it focuses on utilizing the DEMO model in developing an information system. Fig. 1 shows the relationship between the DEMO artifacts and those of the information system. The DEMO artifacts are its four aspect diagrams: the Construction Model (CM), Fact Model (FM), Process Model (PM) and Action Model (AM). The artifacts for the IS used here are use case diagrams and scenarios, activity diagrams and entity relationship diagrams. The CM contributes directly to the derivation of use cases. For each transaction, there will be a group of use cases that meet the functionality of this transaction. The PM contributes only partially to the derivation of use cases. This is because not all the process steps in the transaction need to have a use case; whether a use case is needed depends on the requirements that specify how the users will interact with the IS.

Both the PM and the AM contribute toward drawing needed activity diagrams. There is no need to draw activity diagrams for all the cases. We draw them only where it is necessary to illustrate a particularly important

series of activities.

The AM presents the pre and post conditions for each act. Therefore, it contributes to the use case scenarios. The FM represents the objects that are used in the system; therefore, ERDs (Entity Relationship Diagram) are derived directly from it.

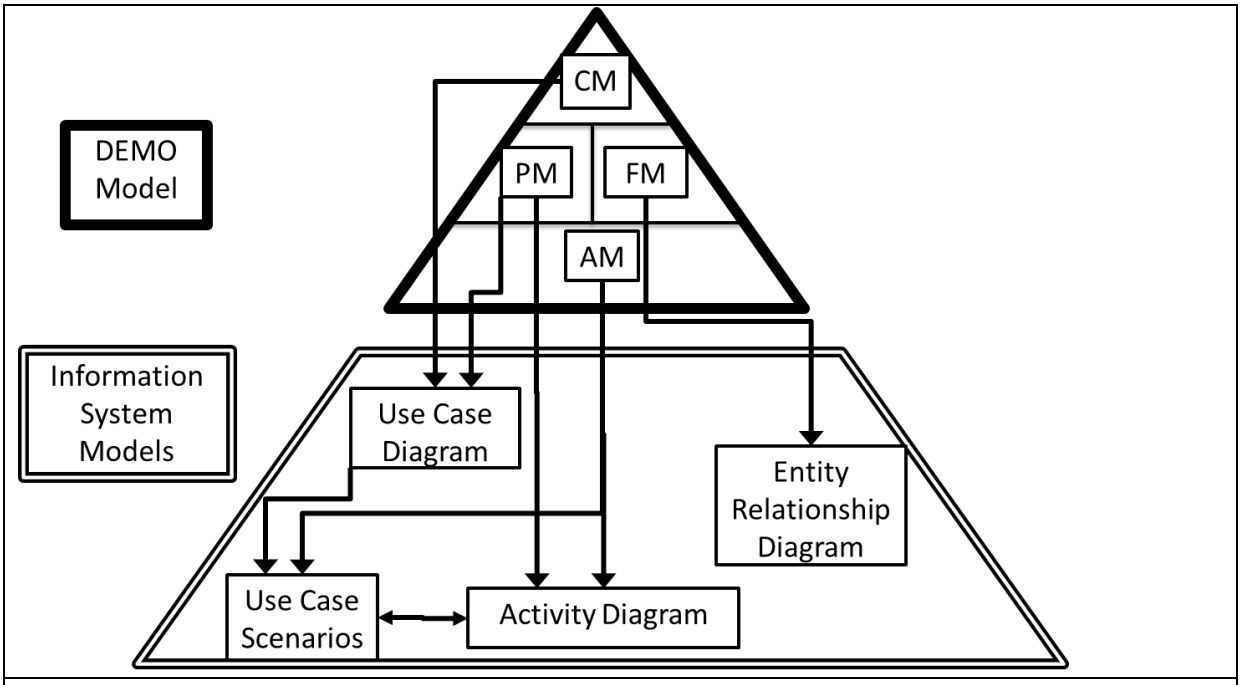

Fig. 1 : Deriving IS artifacts from DEMO

Fig. 2 illustrates our approach by presenting the various stages involved, starting by investigating the scope of the development project. At this stage, the questions to answer are: “What is the scope of the system?” and “What are the boundaries of the system?”

The second stage involves analyzing the users of the system. Who are the actors in this system? What are their roles in the system? These questions will be answered by analyzing the actor roles of DEMO and matching them with the actors of the enterprise. This analysis is called Actor Role-Actor analysis. Any delegation identified should be specified at this stage. Transaction patterns could be used as a reference to identify the possible process steps and who will act on them.

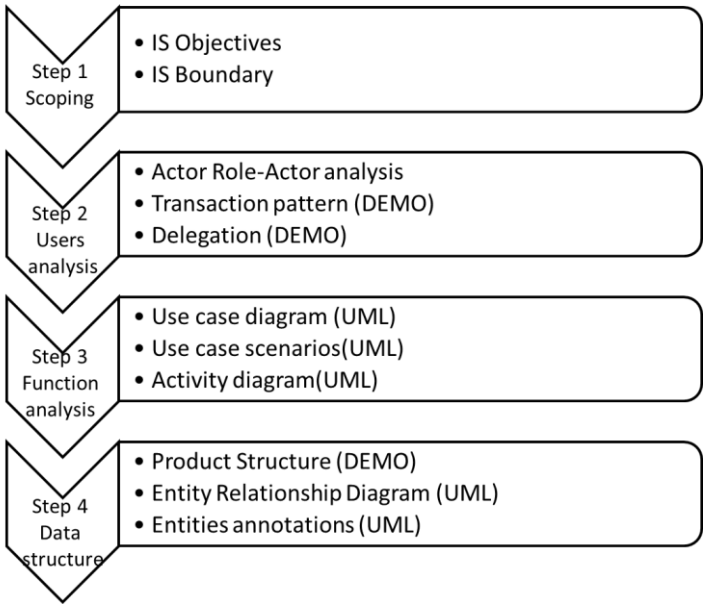

Fig. 2 : DEMO based ISD approach

Third, the functional requirements are specified by deriving the use cases from the transactions. The details of each use case are specified by its scenario. We may draw an activity diagram for difficult use cases.

Fourth, the data structure is developed. The Product Structure shows the entire image of the structure of the system and represents the relationships between the transactions. Then, an Entity Relationship Diagram (ERD) will be developed based on the fact model. The ERD is then expanded by defining the entities and their attributes through entity annotations. The steps are explained in detail as follows:

- **Step 1: Scoping**
  - *IS Objectives*: Setting the objectives of developing the information system must be completed first of all. We must know what the software is expected to do. Why do we need this? It is a common mistake to think that the objective is to automate every process in the enterprise that can be automated. Instead, we must set the objectives based on needs, budgets and time and other resource limitations. One important point in this step is to make sure that the DEMO model covers all the transactions that are related to the objectives.
  - *IS Boundaries*: Boundaries are defined as the limits of the system, where the system meets the environment. Defining the boundary is important in ISD so that developers know what functions they must include and what not to include. Using the DEMO model, it is easy to define the boundary; we simply need to specify which transactions are included in our system and which transactions are out of scope. The boundary should be precisely defined based on the objectives set in the previous step.
- **Step 2: User analysis**
  - *Actor Role-Actor analysis*: DEMO provides the elementary actor roles. However, during the implementation, real users fulfill these actor roles based on the business context. An analysis of who will fulfill each actor role is important during implementation. This could be accomplished by

creating an actor role-actor table in which each row contains one actor and the actor roles that he or she fulfills. Generally speaking, one actor fulfills one or more different actor roles; however, sometimes one actor role might be fulfilled by two actors. This must be defined clearly. After implementation, during usage, the actor roles that were assigned to one actor might change. When this occurs, a control panel must be developed to allow redefining the actor roles assigned to each actor. This analysis is critical to define the authentications and permissions in the information system.

- *Transaction patterns*: DEMO methodology provides three patterns of transaction, named basic, standard and complete. The basic transaction pattern includes only the process steps that are in the happy path. Declining an initiator's request or an initiator rejecting the result of a transaction are not included in the basic pattern. The basic pattern is useful when the execution path is clear and there is no possibility for disagreement between the initiator and the executor. When a possibility for declining the request or rejecting the result of the transaction exists, the standard pattern should be used instead of the basic one. Using the standard pattern, a transaction has three possible outcomes: quit, accept or stop. If revoke capability is required in the IS for a partial transaction, then the complete transaction pattern should be used. This pattern is the most comprehensive and contains all the possible scenarios. Determining which pattern to use for each transaction is based on the business needs analysis. Note that while using the complete transaction pattern for all the transactions might seem like a good idea, doing so makes the IS very complex, and it will contains unneeded scenarios. Therefore, an analysis of which pattern to use must be performed for each transaction.
- *Delegation*: During user actor role analysis, every user will be matched to the actor roles that he or she will fulfill. However, sometimes an actor may delegate his or her actor role to another person. Such delegation might occur for all the process steps of that actor role or it might occur for only one process step, such as a request. Any delegation should be defined clearly in this stage so that the information system implementation will support the delegations.

- **Step 3: Function analysis**
  - *Use case diagrams*: Use case diagrams (UCDs) are popular in ISD because they define the functionality that the information system must provide to the users. A UCD is a graphically based description of the system functions, and it uses natural language. Therefore, it is easy to understand and easy for both domain experts and design experts to validate.

Table 1: Deriving Use Cases from DEMO Transaction

DEMO Transactions	**IS Use Case**
Business (ontological) Transaction (red)	Common Use Cases: 1. Initiating a new transaction 2. Editing an existing transaction

	3. Deleting an existing transaction 4. Viewing the details of an existing transaction 5. Viewing a list of existing transactions Specific Use case: Based on special user requirements.
Infological Transaction (green)	One use case with the same name.
	User-related UCs: 1. Add User 2. View user details 3. Edit user details 4. Delete User 5. View Users list 6. Add role to user 7. Edit user role System UCs: 1. Change password 2. Sign in 3. Sign out

Use cases are not only useful for implementation but are also used for testing the IS after development and for documentation purposes. UCs can be derived from the CM and PM models. When developing UCs from the DEMO model, one should bear in mind that this process is not a one-to-one mapping. It needs analysis and decisions by domain experts and design experts. However, having the DEMO model makes this process easier. Each red transaction needs one or more than one use case. Generally speaking, every red transaction needs one use case to initiate a new artifact, edit an existing artifact, delete an existing artifact, view the details of an existing artifact and view the list of existing artifacts. Depending on the transaction, some may require other use cases, such as calculating derived artifacts. Green transactions do not create a new artifact; therefore, only viewing or calculating use cases can be derived from green transactions. The derivation process is shown in Table 1. Note that additional use cases are always needed in every IS. The use cases listed in Table 1 are user- and system-related use cases.

- *Use case scenarios*: A use case scenario defines the details of the use case. It defines the pre and post conditions as well as all the possible steps that may occur during execution of the use case. All these details can be derived directly from the AM in DEMO. Not every process step in the DEMO PM is transferred to a use case, only the process-related step action models.
- *Activity diagram*: Activity diagrams show the flow of activities in a use case scenario graphically. An activity diagram is used when the use case scenario is complex and needs to be carefully

validated. When an activity diagram is needed, it can easily be derived from the use case scenario and the AM.

- **Step 4: Data structure**
  - *Product Structure*: A product structure shows the relationship between the products of each transaction. It is a tree based structure that helps in understanding what the root transaction is that triggers others. A product structure is based on the composite axiom. It represents the child-parent relationships between the products of the transactions. Product structures are important for building the entity relationship diagram as well in dividing the system into components.
  - *Entity Relationship Diagram*: An entity relationship diagram defines the entities that are handled by the IS and illustrates how these are structured by defining the relationships between entities. A database system is at the base of a information system. Relational databases are widely used in various areas. The Entity-Relationship Diagram (ERD) is a common technique used to define data structures and for database systems design [37]. A basic ERD can be easily derived from the FM in DEMO by following the guidelines shown in Table 2. Every object in the FM except time-related objects will become an entity in the ERD. Time-related objects become attributes of other entities. The FM contains the passive entities of the system; therefore, to define the active entities, one can derive them from the user actor role analysis as shown in Table 2. All the properties of an object in FM will be attributes of the same entity in the ERD.

Table 2: Deriving Entities from DEMO Objects

Objects in DEMO	**Entities in ERD**
Red Object (not time related)	Entity (Extra entities may be needed based on user requirements)
Red Object (time related)	Entity Attribute
Green Object	_
	Users Roles Roles/Users

  - *Entity annotations*: To make sure that both the domain experts and design experts share the same understanding of the system to be developed, we strongly recommend building a glossary of entity annotations for the terminology used in the ERD. This could be created easily by providing a definition for each entity and each attribute.

IV. Case Study

In this section, we show the development of an IS for a company called SMA based on the proposed framework in the previous section.

- **Background of the Selected Case**:

  SMA is a leading information technology company. Using its global network delivery model, innovation network, and solution accelerators, SMA helps global organizations address their business challenges. SMA offers its customers IT solutions by developing software based on the customer's needs or by providing consultation. SMA is a project-based company. Every project belongs to one client—who may have more than one project with the company. The employees do not form a structure based on an organizational chart; instead, they are flexible based on the projects they are currently working on. This flexibility allows the company to respond quickly to changes in the market. Therefore, each project has one project manager leading a team of developers. The project manager is responsible for planning the project as well as following it to completion. Based on the project, there might be an employee who is responsible for delivering the product to the customer. This employee may also be responsible for handling payments from the customer. Otherwise, the project manager delivers and receives payments from the customer. When a new project arrives, the project manager begins by planning the project. The result of this planning is a list of scheduled tasks. After breaking down the project into tasks, the project manager assigns those tasks to the developers. During project execution, new tasks may arise. Therefore, every developer may assign a new task to himself/herself or assign it to other developers. All the tasks must be recorded in the information system to be developed. This is critical for following up on the completion of each task. Moreover, the performance analysis and salary of each employee is based on performance—on the results of completing these tasks. The project manager monitors task completion every week, looking at the completed tasks and the remaining tasks, and reassigning tasks from developers with heavy workloads to those who have less work. This balances the work for every developer and improves project efficiency. At the same time, the manager may control task execution by prioritizing tasks according to their importance. Employees receive their salaries based on their working time. Therefore, they record the time for completing every task they perform. At the end of the month, an accountant calculates the work time for each employee. Each employee has a specific hourly salary rate. Based on this rate, the actual payment is calculated. The salary is the sum of work time multiplied by the salary rate plus any reward. Employees may ask for bonuses or other rewards. These are awarded (or withheld) after evaluating their performance. The project manager analyzes the performance of all employees based on their task completion rate. Moreover, based on the performance analysis, an employees' salary rate may increase or the employee may be given a bonus for a particular project. Employees are free to choose the times at which they work, i.e., day or night, as long as the projects are proceeding as scheduled. This flexibility gives them responsibility for their own time. To keep the level of the skills in the company up to date, the SMA frequently hires new highly qualified developers.

- **SMA DEMO model**:

  Based on the description in the previous paragraph, the DEMO model of SMA can be constructed as follows. Because of space limitations, we focus here only on the organization construction diagram (OCD) of the construction model (CM). The unit of business service of SMA is the delivery of an IT

solution to the customer. Therefore, the first transaction to be identified is the (T1) project completion. The customer (CA1) initiates this transaction by requesting an SMA employee to provide an IT solution. This employee will be called the project completer (A1). A1 initiates four transactions: project fee payment (T2), project planning (T3), project monitoring (T5) and task completion (T4). Note that T4 cannot be requested before the project plan is complete. Because the project manager controls the execution of the plan, he or she takes the role of task manager (A5). This actor initiates and periodically executes (every week) the transaction project monitoring (T5) tasks. Execution of this transaction leads to changes in the project plan. To model the salary and rewards mentioned in the description, salary payment (T6) and reward payment (T8) transactions are needed. To execute T6, we need to calculate the salary. Because there is no original fact in T7 (only calculation), it is an infological transaction (green). To execute T8, we need two sub-transactions: reward decision making (T9) and employee evaluation (T10). To perform the evaluation another calculation transaction is needed (T11). The OCD of the CM is shown in Fig. 3.

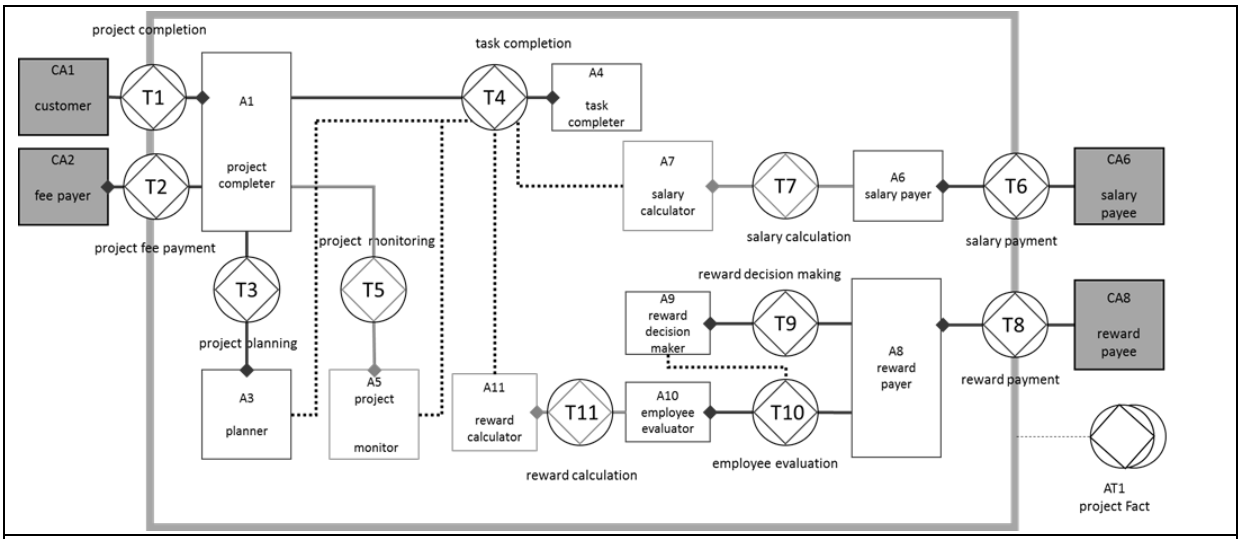

Fig. 3 : OCD for SMA

The CM consists of OCD and TPT. Table 3 shows the TPT for the SMA DEMO model.

Table 3: TPT for SMA DEMO

Transaction	**Product**
T1 project completion	P1 the project is complete
T2 project fee payment	P2 the project fees have been paid
T3 project planning	P3 the project plan has been created
T4 task completion	P4 task is complete
T5 project monitoring	P5 monitoring of Project for Period is complete
T6 salary payment	P6 salary of Employee for Period has been paid
T7 salary calculation	P7 salary of Employee for Period has been calculated

T9 reward decision making	P9 the reward decision of Employee for Period has been made
T8 reward payment	P8 the reward of Employee for Period has been paid
T10 employee evaluation	P10 the evaluation of Reward of Employee for Period has been made
T11 reward calculation	P11 the reward of Employee for Period has been calculated

As we can see from the OCD, there are three main parts: project completion, salary and reward. In the following diagrams, we will illustrate only the reward part due to space limitations. Figure 4 shows the Process Structure Diagram (PSD) of the PM. This diagram shows the sequences during execution of the transactions to determine the reward for each employee.

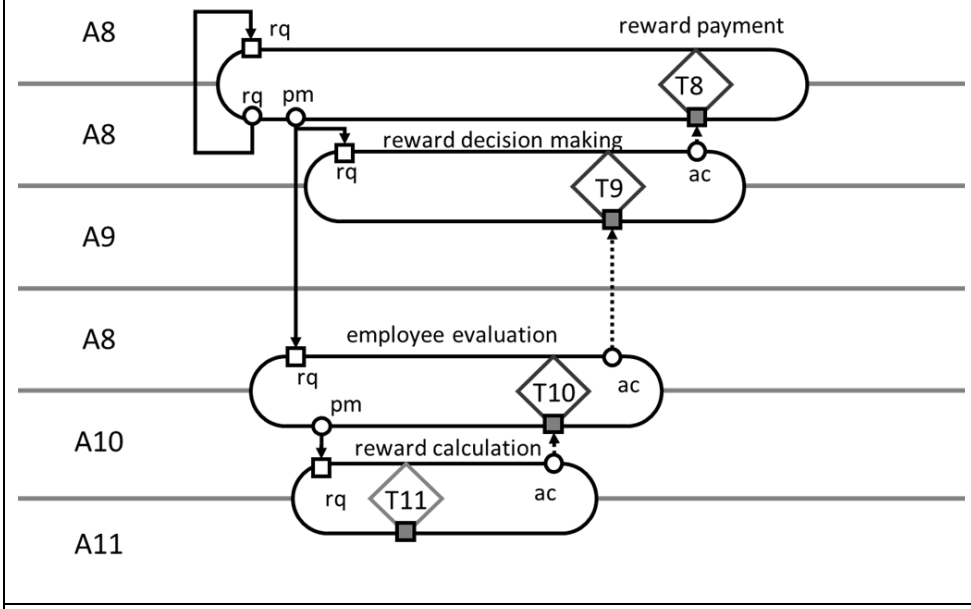

Fig. 4 : A portion of the PSD for SMA

Fig. 5 shows the Object Fact Diagram (OFD) which is part of the FM of the SMA DEMO model. The diagram shows the main objects required for the reward system at SMA.

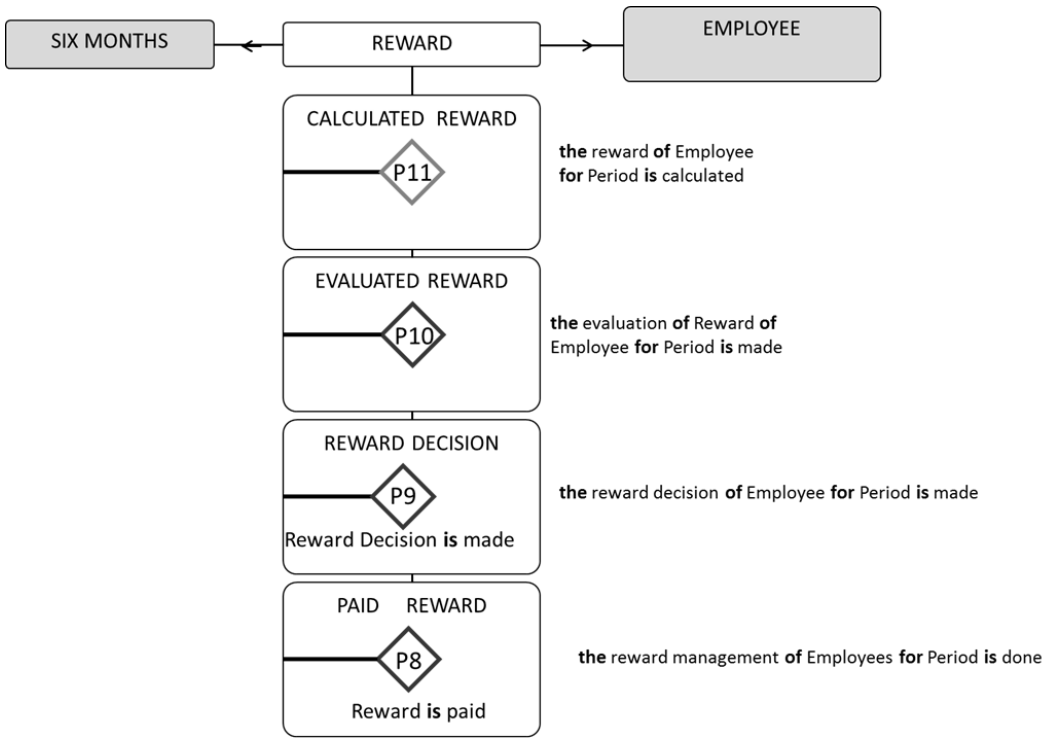

Fig. 5 : A portion of OFD for SMA

The last DEMO model is the Action Model. Figure 6 shows the action rule specification for the first transaction (T1) in the process step request (rq).

Action Rule Specifications for A1 (1)

when new Project completion **is** requested (T1/rq)
with **the** start date **of** Project = <some Day>
the end date **of** Project = <some Day>
the priority **of** Project) **is** <some Priority>
the customer **of** Project **is** <some Customer>
the title **of** Project **is** <some Title>
the description **of** Project **is** <some Description>
if
check justness: performer of [act] **is** authorized **and**
the business context is valid **and**
check sincerity: performer of [act] **is** sincere **and**
check truthfulness: *there is no objection against proceeding*
then project completion **of** Project **must be** promised [T1/pm]

Fig. 6 : A portion of the AM for SMA

The rest of this section will follow the steps in the framework presented in the previous section for developing an IS for SMA.

- **Step 1: Scoping**
  - *IS Objectives*: The information system to be developed has three main objectives. The first objective is to follow up on the execution of all the projects. During project execution, the project

manager needs to know the task statuses and the workload for each employee. This helps to ensure the project meets timeline, quality and cost limits. The second objective is automation. Because the salary of each employee is based on the tasks that are executed by that employee, several calculations are required. To reduce the cost of these calculations, they should be automated. The third objective is the performance analysis. To analyze the performance of each employee, a record of his/her achievements should be archived. Because each task is associated with an execution time, these data can be used to estimate the productivity of the employee.

- *IS Boundaries*: Based on the objectives of the IS, only related transactions are defined in the SMA OCD. This means that transactions related to marketing, recruiting and finance are already excluded. Therefore, SMA OCD includes all the transactions related to the three objectives mentioned in the previous paragraph. However, during the analysis of the required IS with the stakeholders, it becomes apparent that the project fee payment and the project planning steps should not be embedded in the system. The project fee payment transaction occurs only rarely, so it should be a paper based transaction without any relationship to the IS. In contrast, project planning is actually a very complex transaction. And because SMA has adopted Agile methodology, it is very difficult to make a precise plan before beginning to execute the plan. Therefore, it is not required in the IS. The executor of T4 and the initiator of T6 and T8 were intentionally created as external actors. Those actor roles are fulfilled by an employees who are represented as external composite actor roles for two reasons: First; to make it explicit that those employees serve the company through their work, which consists of an accumulation of tasks. In return, they are paid a basic salary and a reward to motivate them to improve their performance. Second, to avoid using the self-initiating transaction type for T6 and T8. Although T6 and T8 are periodically executed without an explicit request from the employee, it is important to show that they should accept the result of this transaction (salary and reward). This reflects their satisfaction just as if they were external customers, and that is why they are always called “internal customers.”

- **Step 2: Users Analysis**
  - *Actor Role-Actor analysis*: After having several discussions with people involved in the project, we agreed on the following table, which represents the relationship between the actor roles and each actor.

Table 4: Relationships between Actors and Actor roles

DEMO actor role	**Actor**
customer, reward manager and reward decision maker	General Manager
project completer, project monitor and employee evaluator	Project Manager (Employee)
salary payee and reward payee	Employee
task completer	Project Member

	(Employee)
salary payer, salary calculator, reward	Accountant (Employee)
fee payer, project planner	Out of scope

Table 4 shows that the customer actor role is fulfilled by the general manager. This is because the real (external) customer has no direct link to the IS. After negotiating with the customer about the project, the general manager decides whether the company will start the project or not. From the table, we can see that the project manager, project member and accountant are employees. This is useful for creating an inheritance relationship among the actors.

- *Transaction pattern*: After having several discussions and analyzing the requirements of the system, we found that the transaction patterns could not be implemented as initially designed. We had to adapt them to the specific needs of the users in SMA. The reason for the change is that some process steps in the transaction pattern are acted upon implicitly by direct discussion, and there is no need to store these communications in the IS. Therefore, we used the transaction pattern from DEMO solely as a reference to analyze the possible process steps required in the IS. Tables 5-7 show which pattern is used as a reference for each transaction. BTP denotes the basic transaction pattern, STP denotes the standard transaction pattern, and CTP denotes the complete transaction pattern.

Table 5: Selecting Transaction patterns

Transaction	**pattern**
T1 project completion	BTP
T2 project fee payment	Out of scope
T3 project planning	Out of scope
T4 task completion	CTP
T5 project monitoring	BTP
T6 salary payment	STP
T7 salary calculation	BTP
T9 reward decision making	BTP
T8 reward payment	STP
T10 employee evaluation	BTP
T11 reward calculation	BTP

As Table 5 shows, T4 task completion is the only transaction that uses the complete transaction pattern. That is because it is the main transaction in this enterprise—all the other transactions depends on its information. Additionally, the T4 transaction has the possibility of a promise or refusal or revocation. The other transactions can use BTP because only the happy path is applicable, and there is no reason to refuse the request or reject the result of the transaction.

- *Delegation*: After having several discussions, we found the following delegations in the SMA

DEMO models.

Table 6: Delegations in SMA DEMO

Transaction	**Act**	**From actor role**	**To actor role**
T4 task completion	Request	Project completer	Task completer
T1 project completion	Promise	Project completer	customer
T6 salary payment	request	Task completer	salary payee
T8 reward payment	request	Task completer	reward payee
T9 reward decision making	Accept	Reward payee	Reward decision maker

From Table 6, the project completer may delegate the request to the task completer to let the employee initiate the task him or herself; however, the acceptance of the result of the task must still be performed by the project completer.

- **Step 3: Function analysis**
  - *Use case diagram*: After selecting the transactions the IS will support, we defined required use cases for each transaction: 80 use cases in total. The following figure shows the derived use cases for the reward-related transactions.

Table 7: Use cases for reward-related transactions in SMA DEMO

Actor role	Actor	Use case	Transaction
reward payer	Accountant	Pay reward	T8 reward payment
reward decision maker	general manager	decide reward	T9 reward decision making
reward payer	accountant	calculate reward for user for period	T11 reward calculation
reward payer	accountant General Manager	View rewards list for user	T8 reward payment
reward payer	accountant General Manager employee	View reward details	T8 reward payment
reward decision maker	General manager	Edit reward decision	T9 reward decision making
reward payer	accountant	Delete reward	T8 reward payment

reward payer	accountant	Edit reward details	T8 reward payment
Employee evaluator	Project manager	Evaluate employee performance for period	T10 employee evaluation
Employee evaluator	Project manager	Evaluate employee task performance	T10 employee evaluation
Employee evaluator	Project manager	View employee evaluation details	T10 employee evaluation

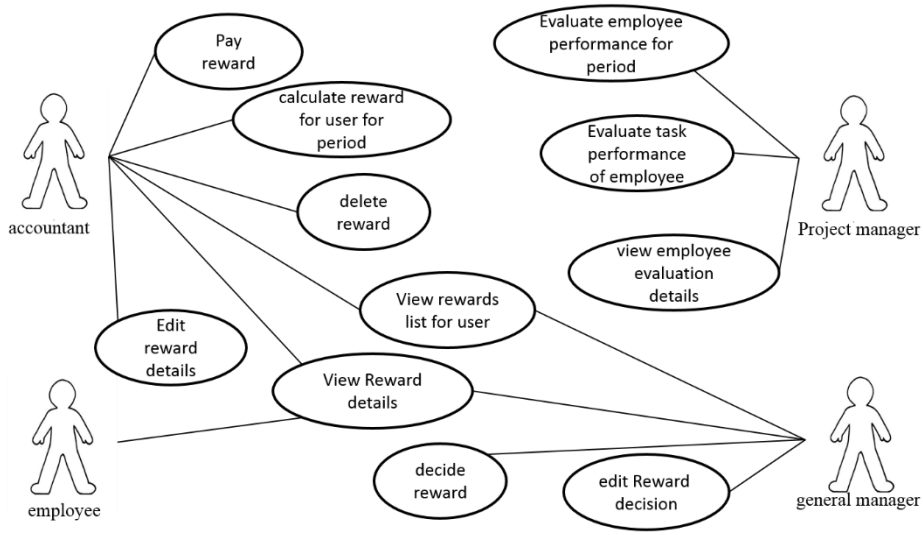

Fig. 7 : Use Case Diagram for Rewards

- *Use case scenarios*: For each use case, we defined the scenarios. Table 8 shows an example for the Add New Project use case.

Table 8: Add New Project Use Case

Use Case Name	Add new project
ID	**UC-01**
Description	This use case shows how a new project can be added to the system.
Actors	General Manager
Pre-condition	User is logged in. User has the permission of General Manager.
Main execution steps	Make request to add a new project Fill in the details of the new project Verify the details of the new project

	Agree on adding the new project with the filled in details
Alternative execution steps	If the project name is empty or a project already exists with the same name, then return to the new project details step. User can cancel the Add New Project operation at any time.
Post conditions	A new project is added

- *Activity diagram*: When the use case scenario is complex, we may analyze it more closely by drawing an activity diagram to illustrate the activities involved in the use case.

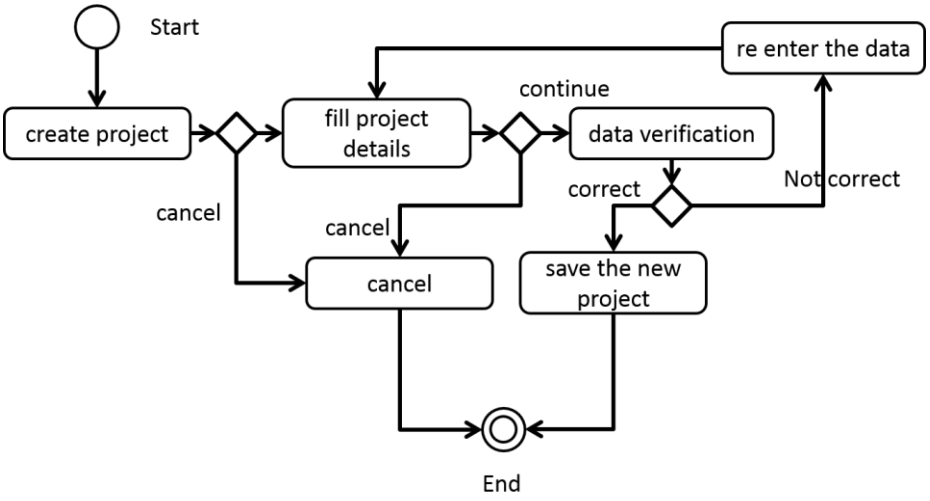

Fig. 8 : Activity diagram for use case Add New Project

- **Step 4: Data structure**:
  - *Product Structure*: The structure of the products of the transaction can be made directly from the OCD. Figure 8 shows the product structure for SMA.

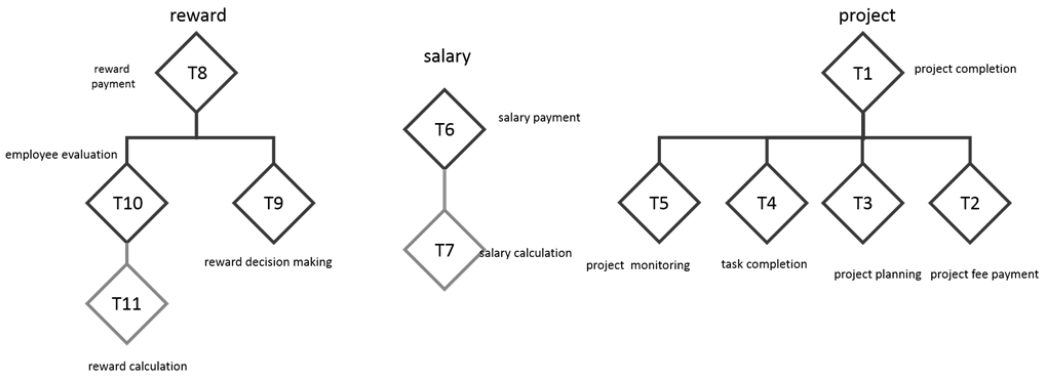

Fig. 9 : Product Structure of SMA DEMO model

From Figure 9, there are three main parts: reward, salary and project. Those three parts are directly related to the objectives of the IS to be implemented.

- *Entity Relationship Diagram*: Deriving an ERD from the OFD of DEMO is an almost one-to-one operation. Each object from DEMO is replaced by an entity in ERD. Table 9 shows the objects from

DEMO and their replacement entities in the ERD. From the table, note that PLAN and FEES were disregarded by the stockholders because they do not want the system to capture those two transactions in the IS. The MONITORING object is a product of a green transaction, meaning it is a derived fact, not an original fact. Therefore, there is no corresponding entity for it in the ERD. The MONTH, PERIOD and SIX MONTHS objects are time-related objects, which are transferred to the ERD as entity attributes. In the T4 task completion transaction, because we adopted the complete transaction pattern, we need additional entities to store the required information. Acts_Tasks stores all the acts related to a particular task. Similarly, Task_Delegations stores all the delegations for the initiating or executing actor role of a task. Task_Comments stores user comments on a task, providing a space for collaboration and discussion between the project team members. Acts_Lists is the set of all possible acts according to the complete transaction pattern. Revoke_Acts stores information about revoked acts. Finally, Task_Execution_Times stores the execution time for each task.

Table 9: Matching Objects of the FM from DEMO to Entities in the ERD

Object from DEMO	**Entities in ERD**	**comment**
PROJECT	Projects	
PLAN	Out of scope	
TASK	Tasks Acts_Tasks Task_Delegations Task_Comments Acts_Lists Revoke_Acts Task_Execution_Times	
MONITORING	_	No original fact, only derived facts.
SALARY	Salaries	
FEES	Out of scope	
MONTH	_	Attribute, not entity
PERIOD	_	Attribute, not entity
SIX MONTHS	_	Attribute not entity
EMPLOYEE	Users	OFD does not capture active entities

	Roles Roles_Users_Projects	
REWARD	Rewards	

- *Entity annotations*: Each term in the ERD must be clearly defined. Table 10 shows a glossary example for the reward entity attributes.

Table 10: Entities annotations of reward entity

Attribute	Type	Description
Id	Integer	Identifier defines each reward.
User_Id	Integer	The user who will receive the reward.
Amount	Integer	The amount of money to be paid as a reward.
Start_Period	datetime	Each reward is paid for a specific period of time. Start_Period defines the start time for the reward.
End_Period	datetime	Each reward is paid for a specific period of time. End_Period defines the end time for the reward.
Description	text	A description by the general manager explaining the reason for this reward.

V. Discussion

In this section, we will discuss the main benefits of our approach to developing information systems and the insights we obtained from applying this approach to one real-world case study. Based on the study, we found that DEMO contributes to information system development in many different ways, the most important of which are analyzing and specifying requirements. Because the DEMO model is concise and simple compared to many other modeling methodologies, it makes the system more understandable by the stakeholders involved in planning the IS. Having a good understanding of the enterprise backed by a solid model such as DEMO makes it easier to specify the requirements of the IS to be developed. In the DEMO model, the ontological transactions are clearly specified. Those transactions represent the functions that the enterprise executes to provide the environment with products and services. And because the actor roles are specified in the DEMO, their responsibilities are clear and available later in the development process. Therefore, there is no conflict in specifying the requirements. And if there are any errors or mistakes in the requirements, it is easy to ensure their correctness by validating them against the DEMO model of the enterprise. This has a great impact on aligning business and IT because DEMO models the enterprise business, and the IS is the way IT supports the business.

DEMO models the business of the enterprise as well as—in an abstract way—the implementation. And the IS supports the business of the enterprise. Therefore, DEMO plays an important role in both creating the roadmap of

the IS and for planning how it might be modified in the future. We can consider the DEMO model as a solid documentation of the enterprise for any further development and maintenance of the IS.

Because DEMO captures all aspects of the enterprise within its four models, it contributes not only to requirements but also to the design of the IS, as shown earlier when we derived the ERD directly from the FM of DEMO. Similarly, the PM helps in modeling activity diagrams for the IS. Finally, the AM plays an important role in specifying the pre and post conditions for each use case.

DEMO's ability to help with IS development increases when many users will interact through the IS. Such IS applications are called interactive information systems [38]. DEMO models are based on modeling the transactions and the actor roles responsible for initiating and executing the transactions (interactivity); therefore, DEMO gains in usefulness as interactivity increases.

In previous research, the infological transaction and actor roles (green) were modeled just like the business ones (red). However, in our research we found there is no need for that. In fact, considering the green transaction the same as the red ones simply makes the model more complex. Green transactions are functions executed by the IS to deliver a particular value to users. In most cases, those functions merely manipulate or display already stored information in response to user needs.

Transaction patterns were used only as a reference when developing the IS. This is because many acts are performed implicitly by direct interaction between people in the enterprise; therefore, there is no need to explicitly store information about these interactions in the IS. In addition, transaction patterns do not consider collaborations between people; for example, some transactions may be initiated or executed collectively by many users, but that is not specified in the transaction patterns.

Finally, self-initiating transactions should be avoided when developing an IS. Transactions are clearer when there are always two actor roles—one responsible for initiating a transaction and a different one responsible for executing the transaction—even though both roles are sometimes filled by one actor. The reason for this is to emphasize that, for each transaction, even those initiated only periodically, someone must be responsible for accepting or rejecting the product of that transaction. Another point we want to make is that the product of a transaction in the FM should not be attributed a time-related object but rather to the real object that includes it. For example, T6 salary payment is a transaction initiated by the employee and executed by the salary payer. The product of that transaction is "The salary of Employee for Period has been paid." Therefore, the product of this transaction belongs to the salary object, not the period object. The period object will be only an attribute of the salary entity in the ERD. If we instead consider that the product is "the salary for Period has been paid," it is misleading; salary is not a product that belongs to the Period object.

Even this research uses a new modeling methodology (DEMO) in modeling the enterprise, yet, in implementation, it uses case diagram, activity diagram, and entity relationship diagram. The later diagrams are well used in software development. Therefore, even the companies who already have their system models can benefit from this methodology. They can benefit from DEMO to check the correctness, consistency, and the comprehensiveness of the current models. This research can contribute to the companies running other models by using our approach, and then compare the resulted use cases and entrees to find out the possible missing use

cases in the current implementation. In addition, the same time, it is possible to find any misalignments in the coordination's acts in the current implementation.

VI. Conclusions and Future Work

In this research we proposed a framework for developing ISs based on DEMO models. The framework derives IS artifacts from the DEMO model. This process is followed by a workflow that provides guidelines on how to implement the framework. The framework emphasizes analyzing the business transactions in the enterprise to derive the functional requirements of the planned IS. Moreover, the framework provides guidelines for deriving the entity relationship diagram from the Fact Model in DEMO. The framework was validated by a real world case study. The results show that DEMO models can play an important role in analyzing and specifying requirements and in designing information systems.

As for future work, there is a need to analyze the derivation of use case diagrams, activity diagrams, and ERDs to make the transformation more straightforward. Finally, developing tools to facilitate and automate the transformation can help organizations that want to adopt this framework.

REFERENCES

[1] E. Yu, P. Giorgini, N. Maiden and J. Mylopoulos, *Social Modeling for Requirements Engineering* (The MIT Press, 2011).
[2] D. Paul and Y.L. Tan, An Investigation Of The Role Of Business Analysts In Is Development, Proc. *Twenty-Third European Conference on Information Systems*, Münster, Germany, 2015, 142.
[3] K. Wright and C. Capps, A survey of information systems development project performance, *Academy of Information & Management Sciences Journal*, 14, 2010, 87-105.
[4] J.L.G. Dietz, System Ontology and its role in Software Development Proc. *Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability*, Rome, Italy, 2005, 8.
[5] T. Gorschek, On the use of software design models in software development practice, *Journal of Systems and Software*, 95, 2014, 176–193.
[6] P.J.M. Mallens, J.L.G. Dietz and B.J Hommes, The Value of Business Process Modeling with DEMO Prior to Information Systems Modeling with UML, Proc. *6th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, Utrecht, The Netherlands, 2001, 31-47.
[7] P. Fitsilis, V.C. Gerogiannis and L. Anthopoulos, Role of unified modelling language in software development in Greece – results from an exploratory study, *ET Software*, 8(4), 2014, 143 -153.
[8] A.M. Fernández-Sáeza, M. Generoa, M.R.V. Chaudronb, D. Caivanoc, and I. Ramosd, Are Forward Designed or Reverse-Engineered UML diagrams more helpful for code maintenance?: A family of experiments, *Information and Software Technology*, 57, 2015, 644–663 .
[9] G. Garousi, V. Garousi-Yusifoğlu, G. Ruhe, J. Zhi, M. Moussavi and B. Smith, Usage and usefulness of technical software documentation : an industrial case study, *Information and Software Technology*, 57, 2015, 664–682.
[10] M. Petre, "No shit" or "Oh, shit!": responses to observations on the use of UML in professional practice, *Software & Systems Modeling*, 13(4), 2014, 1225-1235.
[11] K. J. Letsholo, E. Chioasca and L. Zhao, An Integrative Approach To Support Multi-Perspective Business Process Modeling, *International Journal of Services Computing*, 2(1), 2014, 11-24.
[12] Á. Malta, M. Soares, E. Santos, J. Paes, A. Alencar and J. Castro, iStarTool: Modeling requirements using the i* framework, *Proc. of the 5th International i* Workshop*, Trento, Italy, 2011, 163-165.
[13] T. Fatyani, J. Iijima and J. Park, Comparing DEMO with i_Star In Identifying Software Functional Requirement, *Proc. of the 5th International symposium on Business Modeling and software Design*, Milan, Italy, 2015, 139-149.
[14] J.D. Jong, *A Method for Enterprise Ontology based Design of Enterprise Information Systems*, doctoral diss., delft university of technology, Mekelweg, CD, 2013.
[15] I. Reinhartz-Berger and A. Sturm, Comprehensibility of UML-based software product line specifications, *Empirical Software Engineering*, 19(3), 2012, 678-713.
[16] B. Shishkov and J.L.G. Dietz, Analysis Of Suitability, Appropriateness And Adequacy Of Use Cases Combined With Activity Diagram For Business Systems Modeling, *Proc. Third International Conf. on Enterprise Information Systems*, Setúbal, Portugal, 2001, 854-858.
[17] J.L.G. Dietz, Deriving Use Cases from Business Process Models in Il-Y. Song, S.W. Liddle,T. Ling, P. Scheuermann (Ed.), *Conceptual Modeling* - ER,2813 (Heidelberg: Springer-Verlag Berlin, 2003) 131-143.
[18] P. Bera and J. Evermann, Guidelines for using UML association classes and their effect on domain understanding in requirements engineering, *Requirements Engineering*, 19(1), 2012, 63-80.
[19] F. Alencar, C. Silva, A. Moreira, J. Araújo and J. Castro, Identifying Candidate Aspects with I-star Approach, *Proc. 5th International Conference on Aspect-Oriented Software Development*, Bonn, Germany, 2006, 4-10.
[20] J. Maz´on, J. Pardillo, and J. Trujillo, A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses, *Proc. ER 2007 Workshops CMLSA, FP-UML, ONISW, QoIS, RIGiM,SeCoGIS*, Auckland, New Zealand, 2007, 255-264.

[21] S. António, J. Araújo, and C. Silva, Adapting the i* Framework for Software Product Lines C.A. Heuser, G. Pernul (Ed.), *Advances in Conceptual Modeling - Challenging Perspectives*, (Heidelberg: Springer-Verlag Berlin, 2009) 286-295.
[22] B. Shishkov and J.L.G. Dietz, Design of Software Applications Using Generic Business Components, *Proc. 37th Hawaii Conf. on System Sciences*, Big Island, HI, 2004, 1-10.
[23] B. Shishkov and J.L.G. Dietz, Deriving Use Cases From Business Processes The advantages of DEMO in O. Camp et al. (eds.), *Enterprise Information Systems* V,3 (Netherlands: Springer Science + Business Media, 2004) 249-257.
[24] B. Shishkov and J.L.G. Dietz, Applying Component-Based Uml-Driven Conceptual Modeling In SDBC, *Proc. 21st International Conf. on Electrical Communications and Computers*, San Andres,Cholula, 2011, 82 - 87.
[25] A. Albani, A. Keiblinger, K. Turowski, and C. Winnewisser, Domain Based Identification and Modelling of Business Component Applicationsin L. Kalinichenko et al. (Eds.), *Advances in Databases and Information Systems*, 2798 (Heidelberg: Springer-Verlag Berlin, 2003) 30–45.
[26] A. Albani and J.L.G. Dietz, Identifying Business Components on the basis of an Enterprise Ontology, *Proc. first Interop-ESA Conf. on Interoperability of Enterprise Software and Applications*, Geneva, SW, 2005, 359-370.
[27] L. Terlouw and A. Albani, An Enterprise Ontology-Based Approach to Service Specification Services Computing, *IEEE Transactions on*, 6(1), 2013, 89-101.
[28] M.R. Krouwel and M. O. Land, Combining DEMO and Normalized Systems for Developing Agile Enterprise Information Systems, *Proc. first Enterprise Engineering Working Conf.*, Antwerp, Belgium, 2011, 31-45.
[29] A. Albani and J.L.G. Dietz, Enterprise ontology based development of information systems, *Internet and Enterprise Management*, 7(1), 2011, 41-63.
[30] S. Guerreiro, S.v. Kervel and E. Babkin, Towards Devising an Architectural Framework for Enterprise Operating Systems, *Proc. 8th International Conference on Software Paradigm Trends*, Reykjavik, Iceland, 2013, 578-586.
[31] J.d. Jong, Designing the Information Organization from Ontological Perspective in A. Albani, J.L.G. Dietz, and J. Verelst (Eds.), *Advances in Enterprise Engineering* V,79 (Heidelberg: Springer-Verlag Berlin, 2011) 1-15.
[32] B.J. Williams and J. C. Carver, Examination of the software architecture change characterization scheme using three empirical studies, *Empirical Software Engineering*, 19(3), 2014, 419-464.
[33] M. Dumay, J.L.G Dietz and H. Mulder, Evaluation of DEMO and the Language/Action Perspective after 10 years of experience, *Proc. 10th Anniversary International Working Conf. on The Language-Action Perspective on Communication Modelling*, Kiruna, SE, 2005, 77-105.
[34] Antonia Albani, Jan L.G. Dietz, The Benefit of Enterprise Ontology in Identifying Business Components D. Avison, S. Elliot, J. Krogstie, J. Pries-Heje (Ed.), *The Past and Future of Information Systems: 1976–2006 and Beyond*,214 (Boston: Springer, 2006) 243-254.
[35] M.O. Land and J.L.G. Dietz, Benefits of Enterprise Ontology in Governing Complex Enterprise Transformations, *Proc. Second Enterprise Engineering Working Conf.*, Delft, NE, 2012, 77-92.
[36] G. SCANNIELLO, On the Impact of UML Analysis Models on Source-Code Comprehensibility and Modifiability, *ACM Transactions on Software Engineering and Methodology*, 23(12,13), 2014, 1-26.
[37] Q. Li and Y. Chen, Entity-Relationship DiagramQ. Li and Y. Chen (Ed.), *Modeling and Analysis of Enterprise and Information Systems*, (Heidelberg: Springer-Verlag Berlin, 2009) 125-139.
[38] P. Valente, *Goals software Construction Process* (VDM Verlag Dr. Müller, 2009).