

ENHANCING THE SYSTEM DYNAMICS MODELING PROCESS WITH A DOMAIN MODELING METHOD

FIONA P. TULINAYO* and PATRICK VAN BOMMEL

*Institute of Computing and Information Sciences
Radboud University Nijmegen Heyendaalseweg 135
6525 AJ Nijmegen, The Netherlands
f.tulinayo@science.ru.nl

H. A. (ERIK) PROPER

*Public Research Centre Henri Tudor
29 Avenue John F. Kennedy Luxembourg-Kirchberg
Luxembourg*

*Institute of Computing and Information Sciences
Radboud University Nijmegen Heyendaalseweg 135
6525 AJ Nijmegen, The Netherlands*

Received 28 October 2011

Accepted 4 September 2013

Published 16 October 2013

Defining complex system dynamics (SD) models in complex organizational settings is hard. This is so because the numbers of variables to consider are many and the question of causation is complicated to untangle. Second, SD models are ambiguous and hard to conceptualize. In this paper, we explore the use of a domain modeling method object-role modeling (ORM) in the process of developing SD models. We do so, because domain modeling methods help to identify relationships among entities within the scope of the problem domain and provide a structural view of the domain. The addition of a domain modeling method to the process of developing SD models is to improve SD model conceptualization, enable transformation and reuse of information plus underpin SD with a domain modeling method that allows creation of database. To realize this, we come up with a procedure in our overall research which we refer to as grounded system dynamics (GSD) a combination of ORM and SD. To reason about the combination of SD with a domain modeling method (ORM), we identify and evaluate relationships between their constructs. Basing on the identified relations, ORM to stock and flow diagram (SFD) steps are defined and applied to a real-life case study national medical stores (NMS) situated in Uganda. On completion, we draw conclusions.

Keywords: System dynamics; domain modeling; object-role modeling.

1. Introduction

System dynamics (SD) is a methodology that has its focus on capturing the structure and behavior of systems composed of interacting feedback loops. It was developed by Jay Forrester in 1961 to handle social-economic problems. A review and

history is given in Ref. 1. SD conceptualizes systems as being made up of complex networks of feedback loops where time delays and nonlinear relationships are important sources of dynamic complexity and policy resistance. It offers a systematic approach of qualitative and quantitative analysis which include mapping systems in terms of feedback loops and then translating these maps into quantitative simulation models.² Thus enabling analysis of policy actions, scenarios and consequences by decision makers. Here, we draw our attention to the quantitative aspect of SD. The quantitative aspect comprises of stock and flow diagrams where input parameters result into simulation. More particularly, we identify the relations between object-role modeling (ORM) and SD, evaluate these relations, define the steps on how to transform an ORM model into a stock and flow diagram (SFD) and apply these steps to a case.

In Refs. 3–6 it is stated that SD as a method has a number of issues, one of the key issues identified is SD model conceptualization. In order to conceptualize a given SD scenario, it is important that the “what”, “how” and “why” questions are well answered. In SD however, concentration is drawn to the “what” of the dynamics behavior but not “how” model elements should be reconfigured to yield a desired behavior.⁷ That is why a similar case or model of reality can be depicted (interpreted) in different ways by different SD modelers. To address the issue of SD model conceptualization, different scholars have put in efforts by advocating for better tools and methods to support SD modeling. For example, in Refs. 5 and 3 they suggest mapping of tools to represent SD structure and in Refs. 8 and 6 they suggest techniques to analyze reference modes. Here, we join scholars like Refs. 9–11 who have devised efforts to support the SD conceptualization process with tools and techniques used in other scientific disciplines. In our case however, we address the issue of SD model conceptualization by using support of a domain modeling method ORM whose focus is on domain conceptualization through data modeling.¹² We do so because domain modeling methods help to identify relationships among entities within the scope of the problem domain and provide a structural view of the domain. Second, they capture a detailed representation of the system in terms of system elements whereby their interactions provide a means to understand a given scenario. Finally, domain methods have a precise and consistent mechanism for conceptualizing reality, based on which a stock and flow diagram can be realized.^{7,13} Thus, having domain modeling work in context of SD model building makes SD models more understandable and well focused.

In particular, we use ORM as an example of a domain modeling language because of its conceptual focus and roots in verbalization, graphical expressiveness and well-defined semantics. The main advantage of improving SD’s conceptual foundation through ORM is that SD models can be more soundly and readily linked to databases. Second, introducing ORM in the SD modeling process provides complementary added value: SD studies the behavior of a system in terms of discrete quantities of things (stocks and flows) while ORM underpins the models in terms of underlying ontology of the domain.¹⁴

1.1. ORM as a domain modeling method

ORM is a method that was originally conceived for data modeling purposes¹² and has its foundations in natural language analysis.¹³ ORM takes a static perspective on the domain in the sense that it aims at capturing the fact types and entity types that play a potentially important role in the (dynamic) domain, while SD takes a dynamic perspective in which the dynamic behavior of the domain is captured. ORM is comparable to entity relationship (ER) diagrams in use.¹⁵ It is however, a fact-oriented approach for modeling information and querying the information content of a business domain at a conceptual level.¹⁶ Fact-orientation means that it includes both types and instances in its models; types are called “fact types”, instances “facts”. Including the instance level is crucial in linking concepts with advanced SD modeling. ORM is claimed to have a graphical constraint notation that is far more expressive for data modeling purposes than unified modeling language (UML) class diagrams or ER diagrams.¹⁶

In our previous studies, we looked at conceptual links between SD and ORM,¹⁴ then mapped ORM and SD elements,¹⁷ investigated the update behavior of SD and ORM¹⁸ and came up with steps on how to transform a causal loop diagram (CLD) model into an ORM model.¹⁹ In this paper however, we aim at: (i) identifying the relations between ORM constructs and SFD constructs and (ii) coming up with steps on how to transform an ORM model in an SFD. In order to arrive at the main focus of our overall research (combining SD with a domain modeling method), we came up with a method ground system dynamics (GSD) (i.e. an artifact of the design science approach). This method is a combination of two methods (ORM and SD) and comprises of two sub-phases. The first phase, entails transforming a CLD into an ORM model¹⁹ and the second phase which is the scope of this paper, entails transforming an ORM model into an SFD model. To achieve the overall aim of the study, we use a design science approach. This is because the design science approach focuses first on clarifying the goals of the artifacts (which in this case is GSD) and then on building and carefully evaluating the utility of the artifacts, and to a lesser degree, their reliability and validity.²⁰ The design science approach further places additional emphasis on the iterative construction and evaluation of artifacts which in our case are: the CLD to ORM steps, ORM to SFD steps and their resulting model(s).

1.2. Logical framework

To position this research, we present a logical framework in Fig. 1 where we show the strength and gaps in both ORM and SD. In this framework we itemize (allocate an identification character) each method for convenience in the explanation. We combine method M-1 with method M-3 to arrive at artifact M-2. The gaps and strengths identified in methods M-1 and M-3 are what we use as a basis for our overall study. In Fig. 1 we further present the importance or achievements of combining M-1 with M-3 in M-2. As an output of M-1, M-2 and M-3 is the main

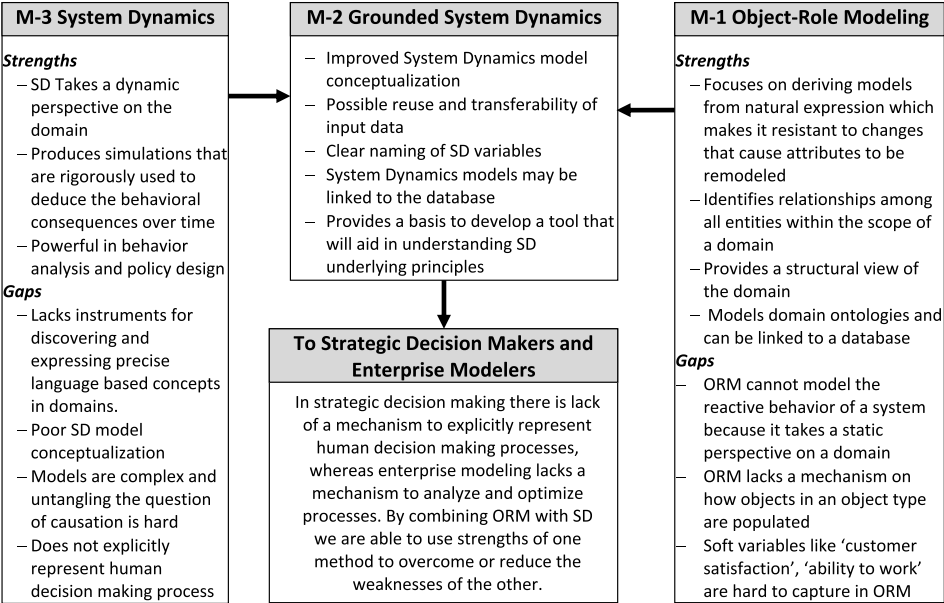


Fig. 1. Research problem and underlying principles.

advantage for having these two methods put to work together in reducing the gap between strategic decision makers and enterprise modelers.

The rest of the paper proceeds as follows: in Secs. 2 and 3 respectively brief introductions to ORM and SD are given. In Sec. 4, we outline the ORM to SFD transformation steps, define and explain both the steps and control parameters. In Sec. 6, we present a snapshot of national medical stores (NMS) from which we derive an ORM model. This ORM model is what we use to apply the ORM to SFD transformation steps. In Sec. 5 we evaluate the ORM to SFD transformation steps and in Sec. 7 we draw conclusions.

2. Brief Introduction to ORM Modeling Constructs

To start with, we present constructs that exist in ORM followed by SD stock and flow diagram constructs. The philosophy behind ORM is that it tries to describe a universe of discourse (UoD) by describing the communication between its members. An ORM scheme basically is a grammar describing that communication. This grammar is also referred to as information grammar. The general construction of an information grammar is as follows. There is a set of syntactic categories (in ORM terminology: object types) and a set of grammar rules (in ORM terminology: fact types) that describe how these syntactic categories are constructed from other syntactic categories. A grammar rule basically indicates what object types

are involved in a fact type and in what role. The term *predicator* is used to indicate such a role. Therefore, in ORM a fact type is seen as a set of predicators. The information grammar describes the elementary sentences that are valid in the associated UoD. From these sentences other sentences may be formed. Object-role calculus (ORC)²¹ and ORM2 (see Ref. 22) are examples of such generic systems for constructing sentences.

ORM's basic building blocks include: entity types, value types and roles.^{23,a}. An *object type* is a collection of objects with similar properties, in the set-theoretical sense. *Objects* are things of interest, they are either instances of entity or value types. Object types are designated by solid-line named ellipses in the graphical reproduction of the information grammar. Some object types have reference modes. These reference modes indicate how a single value relates to that object type. *Value types* on the other hand have instances with a universally understood denotation, and hence require no reference scheme. They are identified solely by their values, their state never changes and they are designated by dotted ellipses. The semantic connections between object types are depicted as combinations of boxes and are called *fact types*. Each box represents a role and is connected to an object type or a value type. The *roles* denote the way entity types participate in that fact type. To represent some of these definitions let us use an example of the procedures a paper might go through en route from writing to publication. The procedures are stated as:

- (1) A person (author) writes intent of submission. This can be in the form of an abstract.
- (2) Then the content (text of the paper) is submitted, whereby the paper becomes a submitted paper.
- (3) Each submitted paper receives a classification.
- (4) Each submitted paper is reviewed
- (5) Some submitted papers are accepted and some are rejected
- (6) For each submitted paper new content is submitted, which makes the paper a published paper that is added to the publications.

This information is represented on an ORM diagram in Fig. 2.

The numerous ORM symbols and constraints used in Fig. 2 can be verbalized as follows:

- (1) (*Exclusive*): **each** paper plays **exactly one of the roles** is rejected or is accepted.
- (2) **each** paper published is **an** accepted paper.
- (3) (*mandatory*): **each** paper is submitted by **at least one** academic.
- (4) (*mandatory*): **each** paper is written by **at least one** academic.

^aFor ORM terminologies in this study, we used Halpin and Morgan (see Ref. 23); and to model ORM models we used natural ORM architect (NORMA).

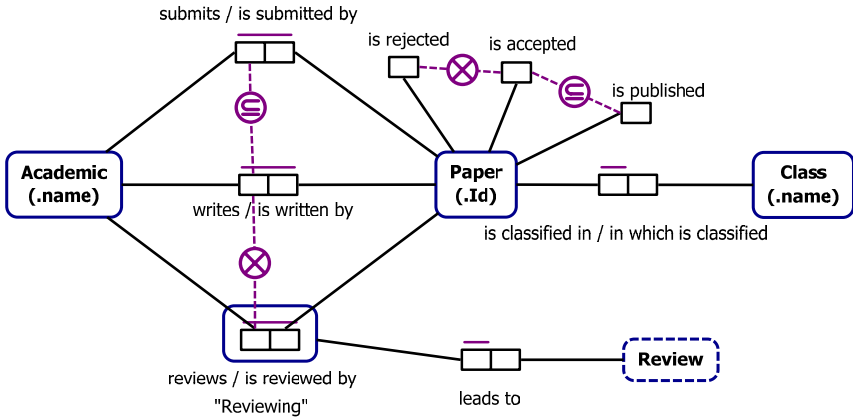


Fig. 2. Paper flow concepts in ORM.

- (5) *each Academic who submits a paper also writes that paper.*
- (6) (*Exclusive*): *each academic plays exactly one of the roles reviews or writes.*

The number of roles in a fact type are referred to as *fact type arity* and the semantics of the fact type are put in a fact predicate. A *predicate* is basically a sentence with object holes in it, one for each role. These predicate names are written beside each role and are read from left to right, or top to bottom. It is through predicates that entity types relate to each other.

Minus the building blocks, ORM also has numerous constraints. Some of these constraints are depicted in Fig. 2 (subset constraint, exclusive constraint, uniqueness constraint and mandatory role constraint). A *mandatory role constraint* (represented as a dot) means that every instance in the population of the role’s object type must play that role. The *uniqueness constraint* (internal) means that instances for that role in the relationship type population must be unique. For example adding a uniqueness constraint over role “is classified” in Fig. 2 means that each paper is classified in at most one class. The *subset constraint* between roles “is accepted” and “is published” means that if some paper is published then that paper is accepted. Here we do not exhaust all ORM constraints but a detailed explanation of these constraints can be found in Ref. 23.

3. Brief Introduction of SFD Constructs

In this section, we only discuss elements that build a SD SFD.^b These elements include: stocks, flows, connectors (information links) and converters (auxiliary variables and constants). *Stocks* are depicted as boxes and are defined as containers

^bFor SD terminologies used in this paper we use Ref. 6 and all SD models are drawn using an SD software called STELLA. This is because it is easy to use and offers a practical way to dynamically visualize and communicate how complex systems and ideas really work.





Stocks	Functions	Examples
Reservoir 	A reservoir is a default stock type. A Reservoir is total number of quantities. It passively accumulates its inflows, minus its outflows at each DT.	Population, water in a tank
Queue 	Think of a <i>Queue</i> as a line of items awaiting entry into some process or activity. Queues are FIFO (first in - first out) in their operation. Stuff enters the queue, and remains in line, waiting its turn to exit the Queue.	grocery store checkout line, airport ticket counter line
Oven 	Think of <i>ovens</i> as batch production system like a bakery process. It opens its doors; fills (either to capacity or until it is time to close the door); bakes its contents for a time (as defined by its outflow logic); then unloads them in an instant.	Elevators depend on door opening or closing to ride and can have a queue waiting to ride in it.
Conveyor 	Think of a <i>Conveyor</i> as a conveyor belt. Material gets on the Conveyor, rides for a period of time (each DT) and then gets off.	Pregnant women, students in school (they ride in a particular class for a period of time and then ride off).

Fig. 3. Forms of stocks, their functions plus their examples.

(reservoirs) containing quantities describing the state of the system. The value of stocks changes overtime through flows (inflows and outflows).⁴ There are different forms stocks can take (reservoir, conveyor, queue, oven), see Table 3. In each of these forms, quantities held are in a different state as explained in Fig. 3.

Flows can be imagined as pipelines with a valve that controls the rate of accumulation to and from the stocks. They are represented as double solid lines with a direction arrow. The arrows indicate the direction of a flow into or from a stock. There exists two types of flows: uniflows and biflows as represented in Fig. 4. A uniflow means that information in that flow moves (flows) in one direction only and the flow takes on non-negative values only. A biflow on the other hand, can take on any value and information flows in two directions. Flows originate from a *source* and terminate in a *sink* which are depicted as clouds.

A *source* represents systems of stocks and rates outside the boundary of the model and a *sink* is where flows terminate outside the system. A sink is located at the arrow tip of the flow and a source is found at the start of the flow arrow.

Converters either represent fixed quantities (constants) or represent variable quantities (auxiliaries). *Auxiliary* variables are informational concepts bearing an independent meaning (add new information). The contained information is in the form of equations or values that can be applied to stocks, flows, and other converters

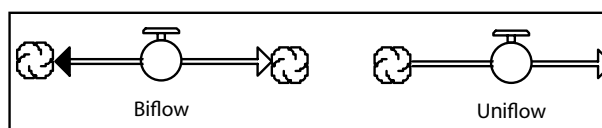


Fig. 4. Types of flows.

in the model.²⁴ *Constants* are state variables which do not change.⁴ Both auxiliary variables and constants are depicted as small circles on the STELLA SD software. Information from converters and flows is shared through *connectors* (information links). Two types of connectors exist, the *action connectors* depicted as solid wires and *information connectors* depicted as dashed wires.²⁵ These connectors are immaterial and connect inputs to decision function of a rate. The underpinned meaning to these connectors is that information about the value at the start of the connector influences information at the arrow tip of that connector. Connectors can feed information into or out of flows and converters but only extract information out of stocks.²⁴ Lastly, we have the concept of *sectors* which are subsystems or subcomponents within a system. They hold/handle all decisions, stocks, information about a particular element or area and contain different information used in an information system. Sectors are not represented in any of the figures but are introduced in one of our later chapters.

Note that among converters we only mention auxiliary and constants but not exogenous variables as building blocks. This is because exogenous variables although they are part of the SFD model, their values are determined by factors outside the model. Second, not all SFD models contain exogenous variables, this means that a model can be complete without any external influence(s).

In conclusion, we present a summary of all the discussed stock and flow building blocks except sectors in Fig. 5 followed by some of the SFD design rules.

Design Rule 1 (Stocks): Each stock should have an inflow attached to it. Through information links, a stock can influence all other variables (converters, exogenous variables or other stocks) but can only be influenced through a flow. In other words, there is no direct connection to a stock other than through flows.

Design Rule 2 (Flows): Every flow is influenced by another variable (stock or converter) in the model through connectors (information links). This enables the

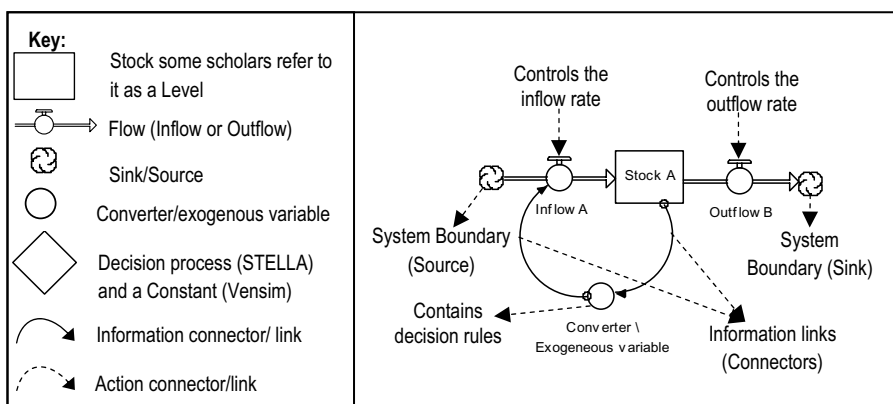


Fig. 5. A summary of SFD basic building blocks.

values in either the inflows or outflows to change the contents in the stock. If there is no variable in the model influencing a flow, then it becomes inactive and the rates in the flows cannot be defined. For a rate to be defined, there must be at least one connector influencing that flow. All in all flows can be influenced by stocks, converters and exogenous variables but cannot directly influence converters and exogenous variables or other flows.

Design Rule 3 (Converters): As we stated earlier there are two types of converters; a constant and an auxiliary. converters should be influenced by at least two or more elements in the model. These elements can either be dynamic or static. Converters and exogenous variables can influence flows or other converters and exogenous variables.

Design Rule 4 (Sink and Source): A sink and source exist on flows that do not originate from or terminate into a stock.

Design Rule 5 (Information links): Information links can feed information into or out of flows, constants, auxiliary variables and exogenous variables but only extract information out of the stock.

4. ORM to SFD Transformation Steps

To reason about the combination of SD with a domain modeling method (ORM), we identify relationships between their constructs. These relations are defined within each ORM to SFD transformation step. To begin with however, we present the initial ORM to SFD transform steps. These steps cover the second phase of the GSD method. In the first phase were CLD to ORM transformation steps and were presented in Ref. 19. The initial steps are then applied using a case and evaluated using questionnaires and a focus group discussion. Basing on the evaluation, we revise ORM to SFD steps and draw conclusions.

4.1. Outline for ORM to SFD transformation steps

To systematically transform an ORM model into an SFD, we come up with initial steps.

- (1) Identify all possible stocks.
- (2) Identify all relevant flows connecting to each stock.
- (3) Identify all possible converters.
- (4) Identify all possible connectors (information links).
- (5) Create sectors.

4.2. Definitions of ORM to SFD transformation steps

As explained earlier, SD has dynamic properties while ORM has static properties. Static properties refer to the possible states of the system under study while

dynamic properties refer to the possible transitions between the states.²⁶ The only way we relate these two methods is by looking at the contents and roles their constructs play in each method and then identify a connection (this connection may not be very concrete because the methods are dissimilar).

Step 1: Identify all possible stocks.

In step one, we identify all possible stocks. As we stated in Sec. 3, a stock is a container representing an accumulation of either a physical or non-physical quantity and is depicted as a box. To relate an ORM element to a stock, we looked for an ORM element with characteristics similar to an SD stock that is; it holds items, accumulates and can be measured. We relate a *stock* to a *unary fact type* (one role) because unary fact types do correspond to properties of entity types (object types) that allow defining of subtypes and they contain properties specific to an object type. To clarify this relation let us take an example of Fig. 6. In this figure, we have object type “*paper*” playing a role *is published* and *is accepted*. Note that objects in an object type do not make a stock but it is the objects in a unary fact type that make a stock. This is because the objects in a unary fact type give an independent state to which some of the objects in an object type take part and in this state they only relate with one object type. This means that all objects in a particular object type are of the same type “*paper*” but in different state *is published* and *is accepted*.

As noted in Fig. 3 of Sec. 3, there are different forms stocks can take (reservoir, conveyor, queue, oven). These forms can be defined in cases were an object type has more than one unary fact type. To identify these forms of stock, we use constraints placed between or among these unary fact types. Note that these unary fact types are attached to the same object type which means that they have similar properties but are in different states (forms). For example, in Fig. 6(a), there are two unary fact types (*is published* and *is accepted*), objects in unary fact type *is published* are a subset of objects in unary fact type *is accepted*. The subset constraint is marked by a dotted arrow running from a subset role to a superset role.

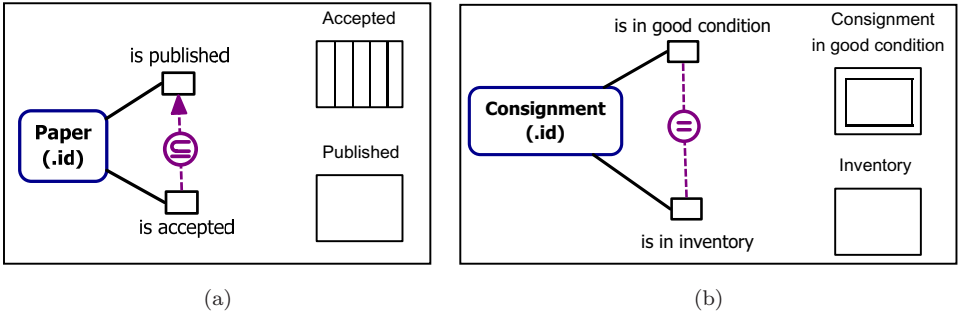


Fig. 6. Examples of ORM subset constraint and equality constraint.

ORM verbalizations for Fig. 6(a) are as follows:

If some Paper *is accepted* then that Paper *is published*. The arrow notion for subset constraints derives from the arrow often used for the logical connective IF.. THEN..

Figure 6(b) Verbalizations

For each Consignment,

that Consignment *is in inventory* if and only if that Consignment *is in good condition*.

In Fig. 6(b), we have the equality constraint placed between roles *is in inventory* and *is in good condition*. This equality constraint indicates that the populations of the role sequences must always be equal. By using the constraints and verbalizations in the ORM figures, we are able to determine the form of stock to use in SD. In Fig. 6(a) unary fact type “*is accepted*” is represented as a *conveyor* stock and unary fact type “*is published*” is represented as a *reservoir* stock because properties of unary fact type “*is accepted*” are assumed to be in that particular state for a period of time, they then get off and are moved to “*is published*” which is a *reservoir* stock well as in Fig. 6b we represent role *is in good condition* as an *oven* stock and *is in inventory* as a *reservoir* stock. This is because properties of role “*is in good condition*” are assumed to be in that particular state for a period of time but they all load at the same time and dispatch to the inventory at the same time (this is noted from the equality constraint used “*if and only if*”) as an *oven* does. In Fig. 6(b), we assume that all consignment that is in good condition is stored in inventory at the same time which is unrealistic because in real life, consignments may be purchased at different time intervals thus a *conveyor*. But we depict both scenarios to show the difference in use of ORM constraints.

To name stocks that result from unary fact types connecting to the same object type, the state change name (unary fact type name) is concatenated with the object type name to give more meaning to the type of quantities flowing into that particular stock (unary fact type). The stock names in Fig. 6 would then be called “*accepted papers*”, “*published papers*” *consignment in good condition* and *consignment in inventory*.

Step 2: Identify all possible flows.

Flows as explained in Sec. 3, are rates of change of a stock representing the inflow and outflow.⁴ They are sometimes referred to as material flows with rates. They are active, change overtime and determine the value of a stock. We relate an *object type* to a *flow*. Object types connect different roles to other object types and objects held by these object types play unique roles for each role connection. That is why we see a similarity with SD flows which connect stocks to other stocks. Second, objects in each object type are similar and unique this also applies to contents in each flow. To clarify this relationship let us use the example given in Fig. 7, in this example we see that *Paper* is the object in object type *Paper*. This therefore means that when creating a flow, it is the content(s) in the object type that flow into stock

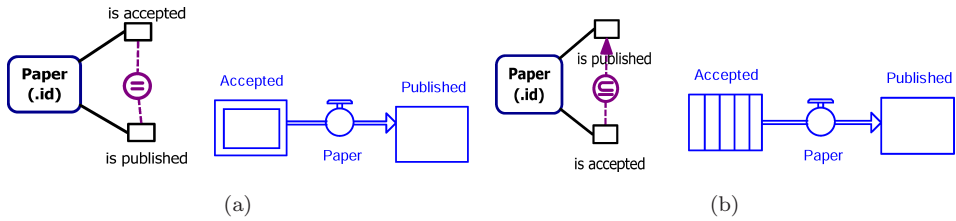


Fig. 7. Examples of a flow added to stocks.

“is published” and “is accepted” (a state to which objects from object type paper have transformed into).

Supertype and subtypes: in ORM it is possible to represent unary fact type *is published* and *is accepted* as subtypes of supertype *paper*. This means we would have three object types (one as a supertype and two as subtype). In this case if we refer to both supertypes and their subtypes as different flows we will have flows containing similar contents but in different states. Yet we know that a specialization relation between a subtype and a supertype implies that instances of a subtype are also instances of a supertype, e.g. in Fig. 8,

$$Pop(Male) \cup Pop(Female) = Pop(patient)$$

This means that we cannot represent both a supertype and subtype as different flows unless if both the supertype and subtype have different roles they play.

In Fig. 8(a) supertype patient has two subtypes (Male and Female). One of the subtypes has unary roles “does not smoke” and “is pregnant” respectively. The ORM verbalizations for this model are as follows:

Each Male is an instance of Patient.
Each Female is an instance of Patient.
If some Female is pregnant then that Female does not smoke.

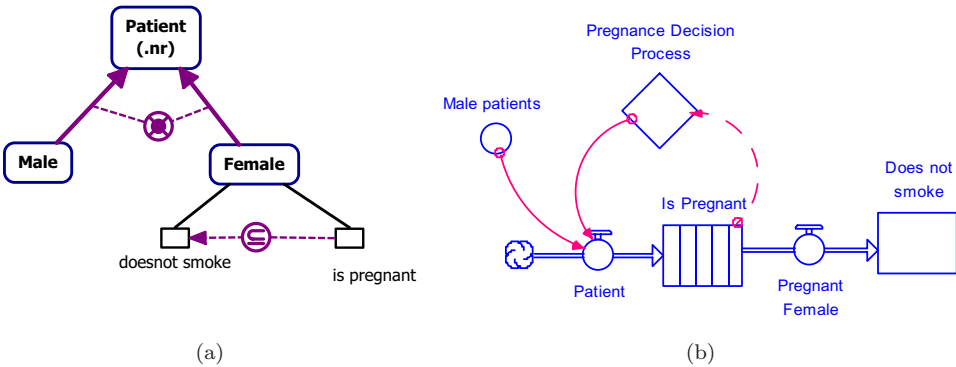


Fig. 8. A supertype-subtype ORM model example plus derived SD model.

Due to the subset constraint between unary role *does not smoke* and *is pregnant*, we use a decision process diamond (DPD) in Fig. 8(b) as a mechanism for managing the diagram complexity associated with the representation of the decision process (*If some Female is pregnant then that Female does not smoke*) within this model.^c As a result, the SD model maintains a bi-focal perspective, displaying the female who are not allowed to smoke because they are pregnant and those who smoke.^d In the SD representation of Fig. 8 instead of having a conveyor, we have a uniflow and a DPD. This is because in stock “*does not smoke*” there are other quantities of persons who may exist but are not pregnant and to determine them are few decisions and parameters need to be input into the model. All these decisions (details) can be captured inside the DPD but if we limit the scope of the model to only pregnant patients, then a conveyor would be appropriate for this particular stock. Another alternative would be to ignore the DPD and instead have the formula for this verbalization (*If some Female is pregnant then that Female does not smoke*) input into the flow. However, this would mean that the effect of this decision would only be seen in the simulation results.

In conclusion, the rates that cause change to an SD stock cannot be represented in ORM. But the object type which is identified to be related to a flow can only help in specifying the contents in a particular flow. Therefore, we suggest that, when deriving an SD model from an ORM model, the new state objects in a unary fact type take should reflect the outflow of that particular stock. For example in Fig. 8 the outflows should be represented as *pregnant female*, *non-smoking female*, etc., that way the modeler is able to relate the type of outflow with the contents (quantities) in the previous stock. Second, there are cases where an object type has no unary fact type but plays a role(s). Here we suggest that, such object types be referred to as converters instead of flows. This is because the objects in this object type are in a constant state but influence fact types (flows and converters in SD).

Step 3: Identify all possible converters.

A converter holds values for constants, defines external inputs to the model, calculates algebraic relationships, and serves as the repository for graphical functions. In general, it converts inputs into outputs. Converters are of two types; auxiliary and constants as stated in Sec. 3. Here we map converters to fact types with more than one role (binary, ternary, etc.). This is because converters add new information to the model and combine two or more variables consistently.^e If fact types have fixed

^cA DPD is a mechanism for managing the diagram complexity associated with the representation of decision processes within a model. Intricacies of decision rules that drive the flows into a “black box” can be “buried” here. On the surface, the modeler and the users of the model can clearly see both the inputs and the outputs associated with a decision process. As a result, the model can maintain a bi-focal perspective, displaying the macro- and micro-structure as needed.

^dIn this study we decline to further discuss or relate DPD to any ORM element due to the complexity attached to it and the fact that some SD software-like Vensim use this same representation as a converter.

^eIn this paper we only consider *binary* and *ternary* fact types to be similar to converters but all fact types with more than one role are referred to as converters.

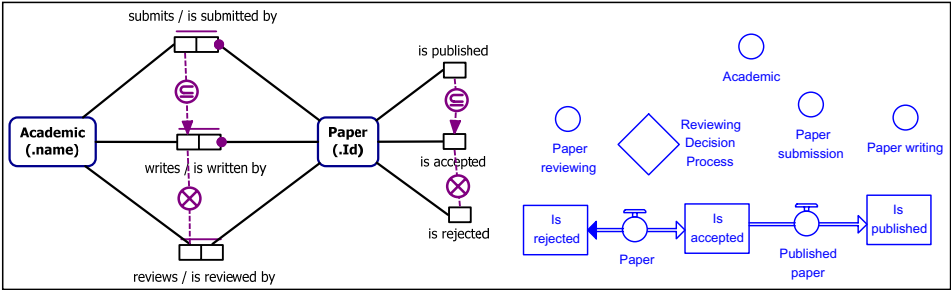


Fig. 9. An ORM and SD model with converters introduced in the model.

objects, then that converter is a constant else it is an auxiliary. Converters vary because they directly or indirectly depend on stocks.

In Fig. 9 we show all converters derived from the ORM model. Note that object academic has no unary roles, therefore we refer to it as a converter as explained in step 2, objects in object type academic play a number of roles (this in SD would be stated as follows; academic influences paper submission, paper writing and paper reviewing) therefore they ought to be included in the model. The only way to include them is by referring to them as converters that influence other converters.

Note that for subset constraint between “is accepted” and “is published” we have a biflow. This is because the reviewing decision brings in either a negative (is rejected) or a positive (is accepted). Since by default it is not possible to have a biflow connected to a conveyor, queue, or oven we do not represent stock (is accepted as a conveyor).

Step 4: Identify all possible information links.

We do not particularly relate *Information links (Connectors)* to any ORM element but instead use the ORM constraints plus verbalizations to identify the direction of the SD connector. In ORM, constraints are placed in between fact types (binary, ternary, etc.), these constraints may play an important role in helping the modeler identify the direction of the connector. Let us use some verbalizations from the ORM model in Fig. 10 to show how ORM verbalizations can help in determining the direction of an information link.

If some Academic submits some Paper then that Academic writes some Paper. (*Connector from paper submissions to paper writing*)

Each Paper is written by some Academic. (*Connector from Academic to paper writing*)

Academic reviews Paper. (*Connector from academic to paper reviewing*)

If some Paper is published then that Paper is accepted. (*Connector from Decision Process Diamond to paper and to published paper*)

However, this mechanism may not exhaustively capture all the relevant connectors, we therefore suggest that the modeler puts the convention to a stock and flow

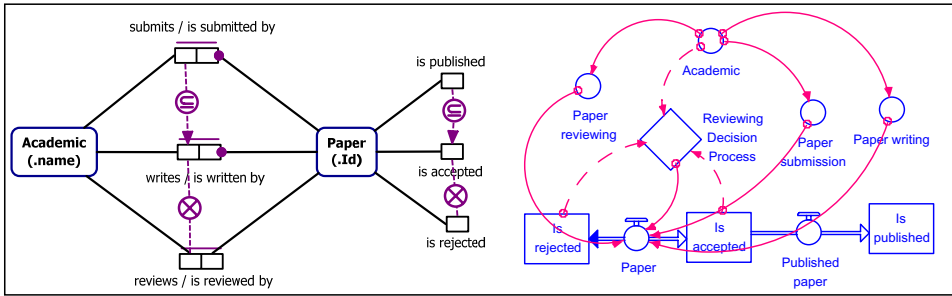


Fig. 10. An ORM and SD model with connectors introduced.

model rule listed below into consideration;

Rule 1: Information links can feed information into or out of flows and converters (auxiliary variables and constants) but only extract information out of the stock.

Rule 2: Stocks are influenced by flows (in and out) and can influence flows or converters but cannot be influenced by other stocks and converters.

Rule 3: The flows can be influenced by stocks and converters but cannot influence converters or other flows, and converters can influence flows or other converters.

Step 5: Identify all possible sectors.

A sector is a grouping of elements with related functionality in a model. For example, in a model of a business organization, you might use a sector to represent each of the major processes under consideration, e.g. a manufacturing sector, a marketing sector, a transportation sector, a human resources sector, and a financial sector in the model. In model analysis, sectors can be run in a sector-by-sector manner. This creates flexibility by enabling sub process analysis.

We relate SD sectors to ORM object types plus their attached roles. This is because ORM conceptual object types act as semantic “glue”^{27,28} and an ORM model is a network of allied object types and relationship types.²⁷ This means that roles and object types when put together, they make up a complete ORM model. Therefore, object types plus their “glued” roles are similar to SD sectors because when both are “glued” or put together they make up a complete model. As explained in Sec. 3, sectors are subcomponents within a system they handle all information about a particular element. If they are subcomponents, this implies that they contain different elements and when these elements are put (“glued”) together they make a complete system. For ORM models with supertypes and subtypes, we map the SD sector to a supertype and not the object types. Therefore a sector in this case comprises of all elements attached to that supertype.

In the Fig. 11 we introduce sectors to the SD model in Fig. 10. There are two sectors derived from the two object type “academic” and “paper”. One of the sectors (academic) has no stock because object type “academic” does not have any unary

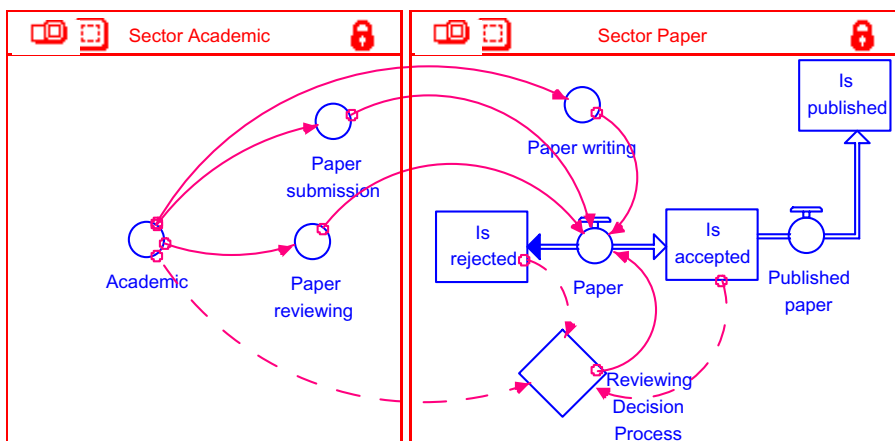


Fig. 11. An SD model with sectors introduced.

fact type attached to it. Second, the converters that relate to both academic and paper can be placed in either of the sectors.

5. Evaluation of ORM to SFD Relations

Evaluation is an iterative process that took place at every stage of the integration. In this section, we present an evaluation for ORM to SD relations. In this evaluation, we used questionnaires and a focus group session. A focus group as defined in Refs. 29 and 30 is a moderated discussion among six to twelve persons discussing a topic under the direction of a moderator whose role is to promote interaction and keep the discussion on the topic of interest. We conducted this evaluation to examine the relations inline with the researcher's pre-knowledge/skills by judging their effectiveness.

In Fig. 12, we present a summary of activities that were used in evaluating ORM to SD relations. Before conducting this focus group discussion, we came up with ORM-SD relations as presented in Sec. 4.2. These relations were used as a basis for the focus group discussion. After defining the initial relations, we prepared documents for evaluating the defined ORM-SD relations. These documents included: defined ORM and SD constructs, defined ORM-SD relations and a questionnaire. When the documents were ready, we identified participants and sent them requests asking them to take part in this focus group discussion. Not all participants replied positively but we received a reasonable number of acceptances for the focus group discussion. Those who responded positively, were asked to suggest dates for the focus group discussion. We schedule the focus group discussion session based on these dates.

Normally questionnaires are given at the end of the discussion. In this case however they were given before and after the focus group session. This was to

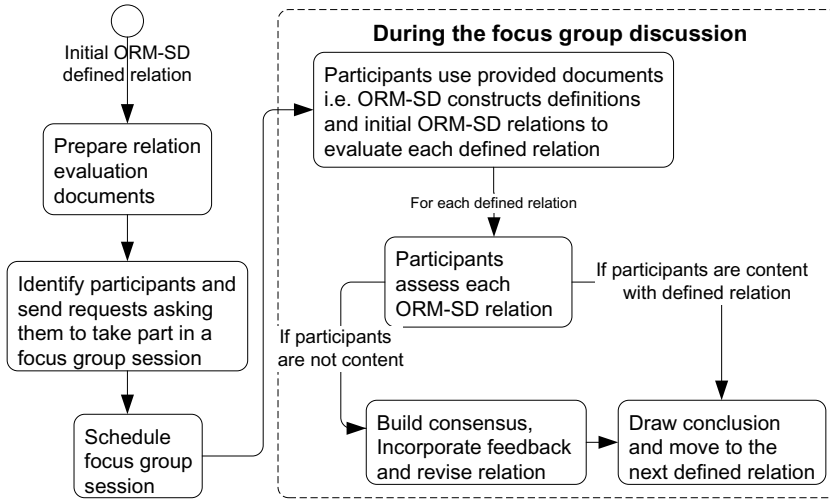


Fig. 12. Summary of ORM-SD focus group session activities.

allow participants to make individual choices of the relations before conducting the focus group discussion. These choices were given to the moderator before the session started to act as a basis of the discussion(s). As each step was discussed, each participant's choice of answer to the relations was loudly readout to participants and discussed accordingly. This way of working made the discussions more interactive and focused. For cases where participants had differing opinion, consensus building was required.³¹ For consensus build, we borrowed a *joint fact finding* approach.³² By borrowing we mean that participants used the idea behind this approach but did not copy or follow this approach step by step. What they did however, was as follows: they gave factual statements that they believed to be relevant to the defined step/relation, exchanged information, developed common assumptions and together used the gathered information to reach a decision. In cases where given facts were not enough or not available, they negotiated a way to find additional information thus filling the gaps or resolving the disagreement(s).

5.1. Analyzing focus group results

To successfully analyze data collected during the focus group discussions, we used the inductive approach. With this approach, analysis is guided by specific objectives and the procedure is systematic.³³ In this study, the analysis phase comprised of a number of activities, e.g. coding, classifying or categorizing of raw data, examining etc. One of the most time consuming activity was, merging of collected data from different sources, i.e. audio recorded data (this had to be transcribed first), questionnaire data (from open ended questions), notes taken by the researcher during the sessions and some comments or remarks some of the participants had put down on their notebooks during the sessions.

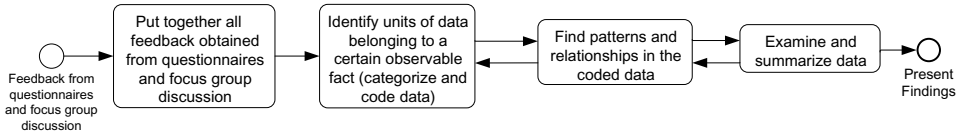


Fig. 13. Summary of how we analyzed focus group data.

As depicted in Fig. 13, we first of all put together all the collected data. This enabled us to systematically carry on our next step which was categorizing and coding. During coding and categorization, we were able to identify portions of data that discussed a particular relation or construct. The different phrases that were found to be pointing to the same ORM–SD relation were given unique codes. As we did so, we were able to collect and analyze facts. This enabled us to find common or different patterns in the text. There were cases when the given text could not fit into the identified relation. In this case, we put our focus on the secluded data to find explanations as to why the given concepts do not fit the patterns. To have a clear portrayal of patterns and relationships, we moved back and forth. That way we were able to find revelations, contradictions and exceptions in the collected data. After examining all the collected data, we drew conclusions and reported the findings.

5.2. Presentation of results

5.2.1. Results from the first questionnaire

Questionnaires were the core of the focus group discussion. In this section, therefore, we present a summary of results obtained from the questionnaire followed by focus group remarks.

In Table 1 we present a summary of the first draft results obtained from the questionnaires. In these results we see that there is no correlation in most of the responses apart from questions 3(a) and 5. Therefore, no conclusive results could be drawn. To improve these results, we conducted a focus group session with the same participants who took part in answering the questionnaires. However, all participants could not be available for the focus group session therefore original answers for those participants that could not attend were kept intact. We received nine answered questionnaires but only seven participants took part in the focus group discussion.

5.2.2. Focus group discussion

In this subsection, we present views or feedback obtained from participants that took part in this study. Before starting the focus group discussion, previously answered questionnaires were given back to participants. This was to enable them to have a clear view of what their previous responses were. During the discussions participants were allowed to make changes to their answered questionnaires.

Table 1. Results from relations questionnaire.

Mapping Questions		Provided Choices							
		Value type	Object type	Subtype	Supertype	Unary type	Binary fact type	Ternary fact type	Constraints
1	Which of the following ORM element is most similar to a system dynamics stock?	0	4	0	1	3	0	0	1
2	Which of the following ORM element is most similar to a system dynamics flow?	1	2	0	0	3	3	0	0
3	There are two SD elements that are classified within a converter; a constant and an auxiliary variable.								
(a)	Which of the following ORM element is most similar to a system dynamics constant?	7	1	0	0	0	0	0	1
(b)	Which of the following ORM element is most similar to a system dynamics auxiliary variable?	3	0	0	1	0	2	2	1
4	Which of the following ORM element is most similar to a system dynamics Information link?	0	0	0	0	2	1	3	3
5	Which of the following ORM elements is most similar to a system dynamics sector?	0	0	2	2	0	0	0	5

To identify a stock from a flow, a mental exercise of freezing the system is normally done. For example freeze a university, walk around what you still see say people, buildings, etc. are the *stocks*. Anything one sees when time is stopped is a stock and anything that can only be seen over a moment of time is a *flow*. Looking at the examples given [see Fig. 10(a)], the number of authors has an influence on the number of papers submitted. In other cases there must be one author that submits the paper, therefore it is appropriate to relate a stock to a role because for the contents in a stock to accumulate, there must be a role played (flow/action) and it is the contents in that act/role that make a stock. Second, the time constant chosen for a variable also determines its representation, e.g. a variable may be referred to as a stock if the time constant chosen is in years and that same variable when its time constant is measured in minutes, it becomes a flow. Considering the definition of object types, they also have some characteristics of a stock. Therefore, both a unary fact type and an object type are related to a stock. If the model has multiple object types in step 5, the model would become too crowded therefore we suggest subtyping as a solution to multiple sectors and stocks.

Flows are identified to have a relation with object types. But as the definition suggests, flows express a dynamic behavior, whereas object types are static. The static/dynamic (behavior) distinction is not discussed in this paper because flows as used in earlier examples represent an anti-rigid type or a dynamic subtype in the literature.^{34,35} This idea of anti-rigid types has also been adopted in more recent works on ORM.³⁶ So, if we take the example of Fig. 10(a), “is published” can be represented through a subtype of *paper* which is *Paper-is-published*? The same can be argued for *is accepted* and *is rejected* as an anti-rigid (dynamic) subtype of *Paper*. The obvious advantages of representing these unary predicates which seem to represent dynamic subtypes is that in this way one could create a taxonomy of those subtypes, ascribe properties that entities have only when instantiating those subtypes (e.g. paper can have properties which are specific for when they are accepted, published or rejected). Note that if a clear real-world semantics for the elements in the language being used is not provided then there is nothing guiding the modeler to choose a modeling primitive over the other (e.g. choose modeling *Paper-is-Published* as a subtype of *Paper* in contrast with the unary predicate). One of the problems with this is that different ORM modeling primitives shall be mapped to different SD primitives. In ORM every object type has associated operations that change its population. Therefore, equating an object type with a flow seems unrealistic. It should therefore relate to an operation related to the object type but this is not modeled in ORM. We therefore suggest an extension of ORM modeling with a model of dynamic aspects (e.g. similar to methods and state chart diagrams in UML) as an intermediary step. This will enable us to have explicit SD–ORM model relations. In this study though, what we have done is relate object types to flows because of their quantities, being able to connect different roles plus through them different object types can be connected. But what is lacking is a mechanism for a flow rate to either depreciate a stock or increase a stock. Therefore, we cannot say that a purely static method can capture all data of a dynamic method even when there has been an extension of ORM with dynamic properties ORM.^{23,36}

Including intangibles or social variables such as perceived customer satisfaction, location attractiveness, ability to work, etc., and combining them as necessary with hard variables in ORM is a significant challenge. This therefore raises doubts that all cases (with soft and hard variables) can be successfully combined. Contiguous entities like masses, amounts of matter, etc. cannot be represented in an ORM (ER, UML) model unless a specific technical treatment is given to it.³⁷ This is because ORM is used for structural domain modeling; ORM model types as collections of individuals which can be individuated, which are countable and have a definite identity.³⁴ SD on the other hand is typically used to model, well, dynamics of the domain and not necessarily involving the flow of countable discrete entities (that is why converters contain information in form of equations or values that can be applied to stocks, flows, and other converters in the model).

From a full set of variables in an SD model, if a variable is not a stock or a flow then it is a converter. Although, it is fine to relate converters to fact types with more than one role, also think of how exogenous variables would be captured in ORM. Maybe they cannot be captured, if not then this should be clarified by ORM experts. Second, converters (auxiliaries or constants) should be given different variable names. In case the value of the auxiliary is equal to a stock, the name of the converter should be changed otherwise we could have an extra variable with exactly the same input numerical(s) (parameters) as the ones for the stock. However, this occasionally happens, usually when it is represented as a separate auxiliary there is a transformation or operation that is done on the basis of the stock. Note that representation of converters with different SD modeling tool (software) vary since there is no standard representation for them.

To relate information links to any ORM variable is inappropriate. This is because information links define how decisions are made. Representing a decision in ORM may require more than one ORM concept and derivation rules. We therefore suggest that information link relates to none of the ORM concepts.

In conclusion, the primary issue here is that we simply cannot completely identify relations between constructs in both methods because each method has something extra which the other method can partly represent. So it is not translating each detail but getting out the essence from one to the other. Therefore we have to accept that we cannot have a complete definition of relation from one method (ORM) into the other method (SD). Otherwise it would be the same method or it would be such a powerful method such that it has everything in it and ends up being useless because it is so complex.

5.2.3. *Results from the second questionnaire*

Since the focus group discussion was conducted because of the diverse answers received from the questionnaires, the questions in the first questionnaire were therefore the core of the focus group discussion. The purpose of the focus group discussion was for participants to further explain their independent perception, elaborate their choice of answers and to have a consensus for each relation. At the end of each relation discussions, participants were allowed to change their answers if necessary.

In Table 2 we present a revised version of the results after conducting the focus group discussion. For question one, five out of nine participants stated that object types are related to stocks because object types are seen as a collection of objects and four participants said unary fact types are related to stocks because of the clear “object type with one property flavor”. For question two, four out of nine participants said that an object type is related to a flow, two participants said that a flow has a relation with binary and ternary fact types because they contain roles with predicates that describe the relationship between or among objects and three participants said that a flow had no relation to any of the provided options. For part (a)

Table 2. Results obtained after focus group discussion.

Mapping Questions		Provided Choices							
		Value type	Object type	Subtype	Supertype	Unary type	Binary fact type	Ternary fact type	Constraints
1	Which of the following ORM element is most similar to a system dynamics stock?	0	5	0	0	4	0	0	0
2	Which of the following ORM element is most similar to a system dynamics flow?	0	4	0	0	0	2	0	3
3	There are two SD elements that are classified within a converter; a constant and an auxiliary variable.								
	(a) Which of the following ORM element is most similar to a system dynamics constant?	9	0	0	0	0	0	0	0
	(b) Which of the following ORM element is most similar to a system dynamics auxiliary variable?	2	0	0	0	0	4	2	1
4	Which of the following ORM element is most similar to a system dynamics Information link?	0	0	0	0	0	1	3	5
5	Which of the following ORM elements is most similar to a system dynamics sector?	0	0	0	2	0	0	0	7

of question 3, all participants said that a constant has a relation with a value type because the values in a value type do not change. For part (b) of question 3, two out of nine participants said that an auxiliary relates to a value type, four participants said an auxiliary has a relation with fact types that have more than one role, two participants stated that an auxiliary is related to constraints because they determine objects that can take part in the relationship and can assume any value, and one participant said that an auxiliary has no relation with any of the provided choices. For question four, three participants said constraints were related to information links because constraints hold some “decision functions”, one participant said binary and ternary fact types relate to information links and five participant said information links have no relation with any of the provided choices. Finally, for question five, two participants said that a sector has a relation with a supertype and seven participants said a sector has no relation with any of the provided choices. During the focus group session there was a lot of arguing which required consensus building and re-explaining of concepts was often required. However, we were able to obtain better or improved results with explanations.

5.3. Revised ORM to SD relations

Considering the remarks presented in Sec. 5.2, we now present the revised ORM to SD relations in Table 3. In this table we relate a stock to two ORM elements (a unary fact type and an object type). *Stocks* are related to ORM *unary fact types* because they both uniquely hold quantities with similar properties and relate to one element (object type for ORM unary fact types and flows for SD stocks). We refer to their elements as *containers* because of their purpose which is holding items.

For the relation between a flow and an object type we decided to keep it as originally defined since participants neither agreed nor rejected relating flows to object types. Keeping in mind that the operation that change contents in an object type is not represented in ORM. Thus, the identified relation between *flows* (inflows and out flows) and *object types* is due to the fact that both constructs hold similar contents that change states, e.g. from a similar object type, objects contained in each unary fact type are in different states and flow quantities change state/differ for each flow-stock connection. Furthermore they all connect different stocks (SD)/unary fact types (ORM) and transfer objects (ORM)/quantities (SD). We refer to their elements as *homogeneous connectors* because they all connect and transfer quantities with similar properties or concepts. As noted by participants, object types also have some characteristics of a stock. Basing on this remark, we came up with two choices of option. These choices are to guide modelers on when to use object types and unary fact types as stocks or flows. The two options are *progressive refinement* and *pure mapping*. Before defining the two options, let us first define the control parameters. These control parameters show all possible choices of option when deriving an SD model from an ORM model. The control parameters include: *RolesObjectStock* (*ROS*) and *UnaryStock* (*US*) in steps 1 and *SectorObject* (*SO*) and *NoSectorObject* (*NSO*) in step 5. In steps 2–4 we do not have any control parameters. Figure 14 presents a summary of all control parameters plus choices of options identified in the transformation of ORM into SFD.

The control parameters in Fig. 11 are explained below, but to start with let us define the elements used where:

$$\begin{aligned} Role &= R_i =: i = 1 \dots n; \\ Objecttype &= O_t =: t = 1 \dots n; \\ Sector &= S_x =: x = 1 \dots n; \end{aligned}$$

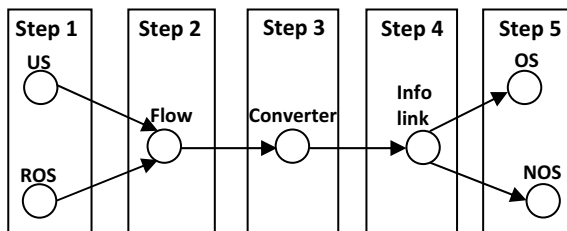


Fig. 14. GSD control parameters.

$Unaryfacttype = U_f =: f = 1 \dots n;$
 $Stock = S.$

- *RoleObjectStock (ROS)*: In this control parameter, all roles and object types are mapped to stocks. Therefore, the total number of stocks in *ROS* is equal to the total number of roles plus the total number of object types in a given ORM model. Therefore, $Pop(ROS) = (Pop(R_i)) \cup Pop(O_i)$
- *UnaryStock (US)*: In this control parameter, only unary fact types are mapped to stocks. This is because objects held by a unary fact type relate to one object type. In one to have a stock for all objects in a particular object type, a unary fact type should be connected to it. This unary fact type acts as a container that holds all objects within that object type. Therefore, the total number of stocks in *US* is equal to the total number of unary fact types in a given ORM model. Therefore, $Pop(US) = (Pop(U_f))$
- *SectorObject (SO)*: This control parameter means that an object type plus roles connected to that object type are mapped to a sector. The total number of sectors therefore, is equal to the total number of object types in a given ORM model. Therefore, $Pop(S_x) = pop(O_k)$
- *NoSectorObject (NSO)*: This control parameter means that an object type plus roles connected to that object type are not mapped to a sector. Therefore, there are no sectors introduced in the final SD model. Having defined the control parameters, we now explain the different routes of option that can be followed while deriving an SD model from an ORM model.

Progressive refinement: In this option, a first draft model is created and progressively elements of that model are replaced by others. For example, in Fig. 14, if a modeler opts to use control parameter *ROS* in step 1, the total number of stocks will be equal to the total number of object types plus total number of roles. In the 2nd step, as we stated in step 2 of Sec. 4.2, flows are mapped to object types. Therefore, all stocks that result from object types are eliminated and instead represented as flows. In the 3rd step we have converters introduced, as we stated in step 3 of Sec. 4.2, converters are mapped to binary and ternary fact types. Therefore, all stocks that make a binary or ternary fact type are eliminated and represented as converters. In the 4th step connectors are added and in the 5th step, the modeler may opt for either *SO* or *NSO*. If the modeler opts for *SO*, each object type plus roles connected to that object type make a sector while for *NSO* there is no ORM element identified to relate to a sector therefore, they are not introduced in the model. In summary this route of option is represented in Fig. 15.

Pure mapping: Here elements are generated only when one is sure that they are the right ones. For example, in Fig. 14, when a modeler opts to use control parameter *US* in step 1, the total number of stocks will be equal to the total number of unary fact types. In the 2nd step, flows are mapped to object types. Therefore, the total number of flows is equal to the total number of object types. In this step, each derived flow is connected to the stock it relates with. In the 3rd step we will have

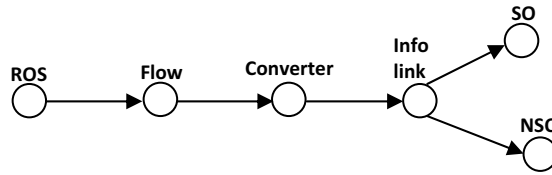


Fig. 15. Progressive refinement option.

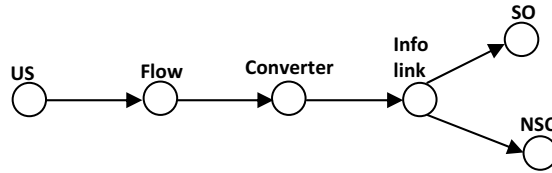


Fig. 16. Pure mapping option.

converters introduced, as we stated in step 3 of Sec. 4.2, converters are similar to binary and ternary fact types. In the 4th step, connectors are introduced. In the 5th step NSO, the modeler may opt for either SO or NSO. If the modeler opts for SO, each object type plus roles connected to that object type make a sector while for NSO sectors are not mapped to any SD element therefore, they are not introduced in the model. In summary this route of option is represented in Fig. 16.

Furthermore, SD *quantities* are related to ORM *counting objects* because they are looked at as quantities that flow within the system or process. We use the term “quantity” to represent items or quantifications that flow within an SD system and their elements are referred to as *contents*.

Converters are related to fact types with more than one role (binary, ternary etc.) because converters add new information to the model and combine two or more variables consistently. For fact types with fixed objects and object types with no attached unary fact type, the converters are constant else they are auxiliaries. Converters vary because they directly or indirectly depend on stocks.

Furthermore participants said that, since *information links* define how decisions in the SD model are made therefore they should not relate to any ORM element. Note that we have not included sectors in Table 3 because participants said that they are not very important in the SD model.

Finally, sectors in SD are an informal notation inform of making things easier to understand but there is really no need to have these sectors. Therefore, they are not required in the SD model as such.

In conclusion, there are light spots in both methods and the given examples can be partly translated but we are coming to boarder that there is something extra in ORM and in SD therefore they are not completely overlapping. The purpose of these steps therefore is to define the visual display of the model. If we were to look at the input parameters into the model then we may need more steps because

Table 3. Revised ORM to SD relations.

System Dynamics Construct	Identified ORM Construct Relations	Transitional Statement	Elements
Stock	Unary fact type	They both contain “things” or act as containers.	Container
	Object type	They both hold “things”, connect roles (ORM)/flows (SD) to other object type (ORM)/stocks (SD) and contents in both constructs play unique roles for each connection.	Connector
Quantity	Objects	These can be looked at as the contents within the system.	Content
Flows (Inflow and Outflow) Converter	Object types	They all connect different stocks (SD) and Fact types (ORM).	Homogeneous connectors
	Fact types with more than one role	Converters are related to <i>fact types</i> with more than one role (binary, ternary and quaternary) because they add new information to the model and combine two or more variables consistently. If fact types have <i>fixed objects</i> , then that converter is a constant else it is an auxiliary. Converters vary because they directly or indirectly depend on stocks.	Heterogeneous connectors
Information Link (Connector)	None	Information links are not related to any ORM element but ORM constraints and verbalizations can guide in identifying connectors and the direction of the SD connector. In ORM constraints are placed in between fact types (binary, ternary, etc.), these constraint may play an important role in helping the modeler identify the direction of the connector.	None

the five steps lead to a picture and we need the details (input parameter guide) for example, what does variable *paper* depend on to be *rejected* or *accepted*? And that is more complex as it is but may be that is not information captured in this form but information captured from the domain expert. Second, in ORM we seem not to have flows explicitly and some of the given roles are mutually excluded thus in a stock and flow diagram, we would have a sequence and that is probably the dynamic aspect (time aspect).

6. Application of the ORM to SFD Transformation Steps

Before we proceed with the application of the GSD procedure, let us give a snapshot of the case NMS which we use to come up with an ORM model. Part of this

ORM model is extracted and used to apply the GSD procedure in deriving a SD model.

6.1. National medical stores (Uganda)

NMS is an Ugandan parastatal where medical supplies are stored before they are assembled and distributed to different health centers or hospitals in the country. NMS warehouse is subdivided into zones that have fixed address locations with physical sequentially labeled alphanumeric code. Storage locations are divided into picking faces and bulk storage locations. Pick faces are arranged to be either in the carton flow rack (for 1–10 unit picks) or ground locations near the marshalling area (fastest moving items). During receipt and movement, all stock pallets are placed in already existing locations. In a snapshot, the process is as follows: goods are brought into the warehouse on trucks, they are received by NMS employee(s), quality assurance inspects the goods to find if there is any quantity or quality inconsistency. If goods are in good condition, they are offloaded from the truck(s), scanned, moved to storage location in bulk and inventory system is updated. When an order is received by NMS employee(s), it is given to one of the NMS warehouse employee (inventory manager or store manager) to add it to order backlog. After a period of time, orders are picked and grouped according to customer requests. Upon completion of the picking process, orders are stacked on the right unit load (e.g. a pallet), scanned, labeled and load on trucks for delivery. After loading the trucks one of the employees determines the number of orders processed (orders filled) and delivered. Any unprocessed order remains in the order backlog until the next run. On trucks, goods are assembled in such a way that those going to the last destination are loaded first and those going to the first destination are loaded last. This makes offloading and delivering easy. In Fig. 17, we present an ORM model derived from the case information stated above. In this model we look at the process from when a consignment is delivered to the warehouse up to when it

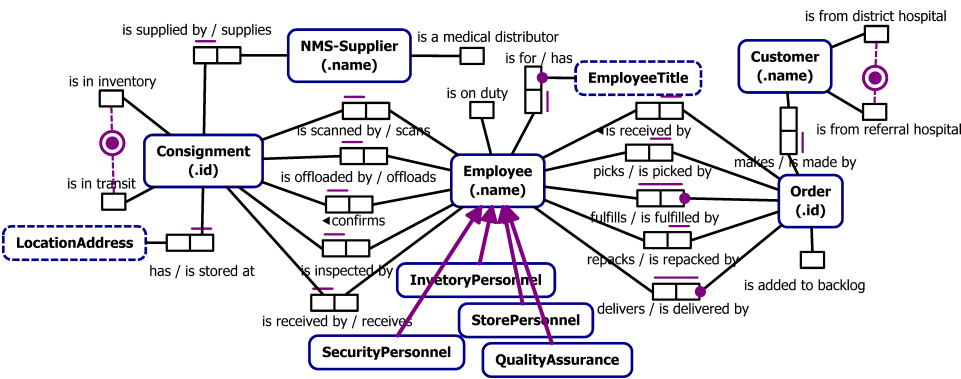


Fig. 17. NMS ORM model.

leaves the warehouse. Note that consignments only go out of the warehouse after an order is placed by a customer.

6.2. Applying ORM to SFD transformation steps

To apply the ORM to SFD steps described in Sec. 4.2, we use the ORM model shown in Fig. 17. In this application procedure, we opt to use the pure mapping option presented in Fig. 16.

Step 1: Identify all possible stocks.

As explained in step 1 of Sec. 4.2, in the pure mapping option, control parameter US. The total number of stocks is equal to the total number of unary fact types. Unary fact types identified are “is on duty”, “is in transit”, “is in inventory”, “is added to backlog”, “is from district hospital”, and “is from referral hospital”. These unary fact types are depicted as boxes because stocks in SD are represented as boxes see Fig. 18.

Step 2: Identify all possible flows connecting to each stock.

As stated in step 2 of Sec. 4.2, object types are mapped to flows because they connect different roles containing different objects. In Fig. 18, we identify all flows connecting to the specified stocks. Identified flows from Fig. 17 include: *are Employee, Order, Consignment, NMS Supplier* and *Customer*. Note that stocks “is from a referral hospital” and “is from a district hospital” plus stocks is in inventory and is in transit have one flow referred to as a biflow. This is because they are of the same object type and have an “or” constraint which indicates that the contents within this flow move in either direction. We therefore make the flow connecting to both stocks a bi-flow (two directions). Second, the quantities (objects) moving through these flows are similar but flow to either stocks. Biflows in SD take on any value and quantities flow in both directions.

The direction of arrows on the flows indicates either an inflow (into the stock) or an outflow (from a stock). In Fig. 19 however, we have no outflows, this may be an indicator that all required flows cannot be directly derived from a given ORM model. Second, in Fig. 19 we also introduce DPDs. This is because they

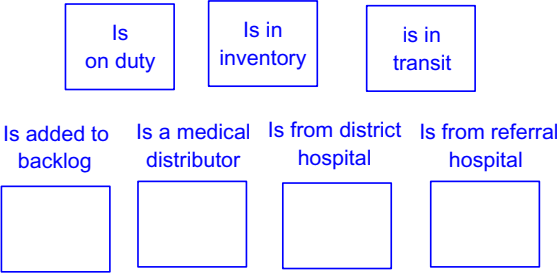


Fig. 18. Unary fact types represented as stocks.

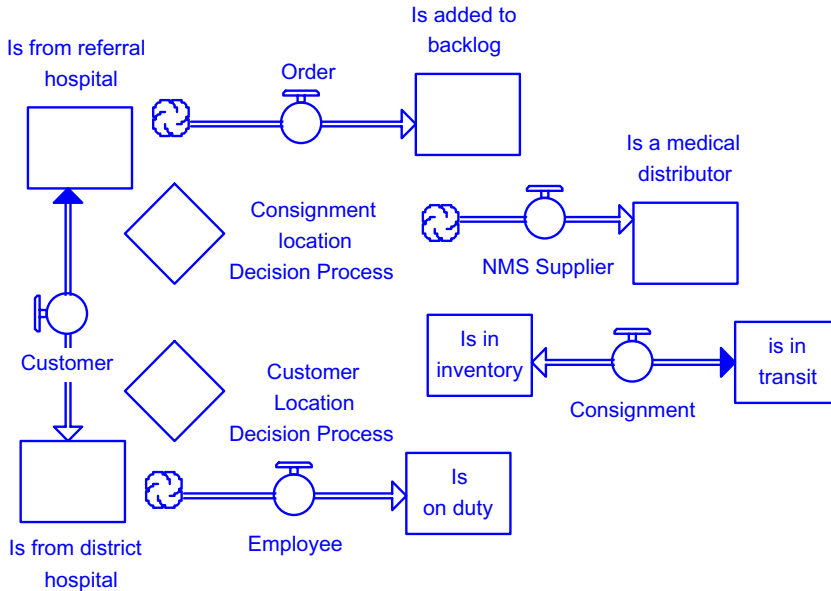


Fig. 19. Flows derived from object types.

help in managing the diagram complexity associated with the representation of decision processes for example deciding on whether the consignment is in transit or in inventory. These DPDs are applied to ORM model elements that require a decision to be made for example “is from district hospital”/“is from referral hospital” and “is in inventory” and “is in transit”. These elements have exclusive or constraints and exclusion constraints, etc.

Step 3: Identify all possible converters.

As we stated in step 3 of Sec. 4.2, fact types that have more than one role are mapped to converters. From the NMS ORM model given in Fig. 17, we have the following fact types with more than one role (binary and ternary fact types); *is scanned by/scans*, *confirms/is confirmed by*, *picks/is picked by*, *is received by/receives*, *delivers/is delivered by*, *fulfills/is fulfilled by*, *repacks/is repacked by*, *inspects/is inspected by*, *offloads/is offloaded by*, *supplies/is supplied by* and *makes/is made by*. To derive meaningful converter names, we use both the predicate names plus object type names. That is to say, we concatenate the fact type name to the object type name to get the converter name. For example, roles *makes/is made by* we refer to the fact type as “making”, for roles *confirms/is confirmed by* we refer to the fact type as “confirmation”, for roles *is fulfilled by/fulfills* we refer to the fact type as “fulfillment”, for roles *is received by/ receives* we refer to the fact type as “reception”, for roles *delivers/ is delivered by* to we refer to the fact type as “delivery”, for roles *is picked by/ picks* we refer to the fact type as “picking”, for roles *is inspected/ is inspected by* we refer to the fact type as “inspection”, for roles *scans/is scanned*

by we refer to the fact type as “scanning” and roles *offloading/is offloaded by* we refer to the fact type as “offloading”. Having named all fact types, we now concatenate the fact type name to the object type name giving us: *Consignment picking*, *Order making*, *Consignment confirmation*, *Consignment inspection*, *Consignment repacking*, *Consignment scanning*, *Consignment reception*, *Consignment offloading*, *Consignment supply*, *Order delivery*, *Order reception* and *Order fulfillment*. This information is represented in Fig. 20.

Note that in the NMS ORM model shown in Fig. 17, there are value types “*EmployeeTitle*” and “*LocationAddress*”. Value types as explained in Sec. 2 are constants; they are depicted as boxes with dashed lines. They are identified solely by their values and they never change their state (i.e. making them constant). For example: *employee title*, *name*, etc. Since value types are distinguished from each other solely by their values. Therefore, we map them to constants in SD because constants are state variables which do not or change slowly⁴ that they could be assumed constant for the time scope of the model. In Fig. 20 though, we do not include both the value types plus their fact types because we intend to further study the effect of mapping a value type to a constant and how they should be represented in the SD model.

Step 4: Identify all possible connectors (information links).

Connectors as explained in Sec. 3 are immaterial and connect inputs to decision function of a rate. The underpinned meaning to these connectors is that information about the value at the start of the connector influences information at the arrow tip of that connector. Connectors feed information into or out of flows and converters

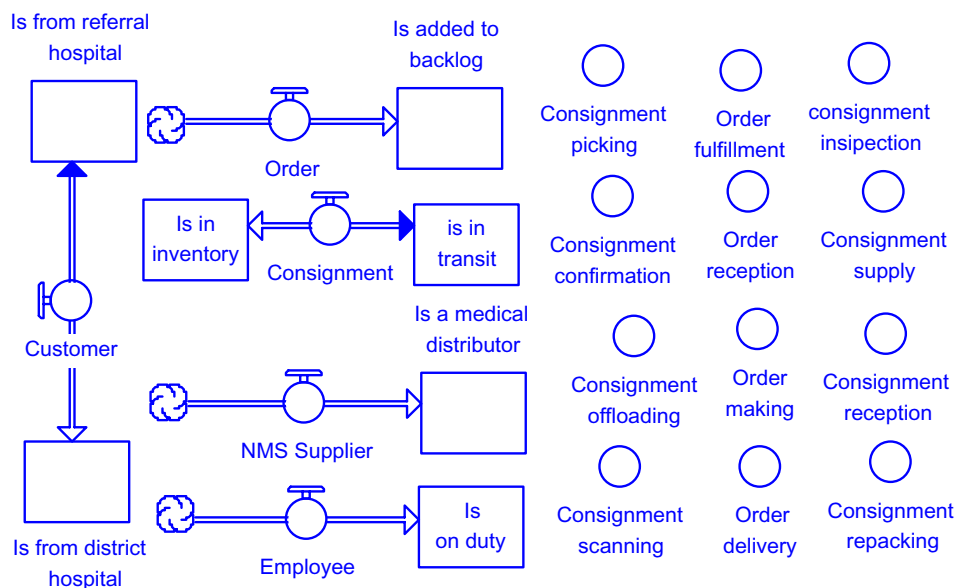


Fig. 20. Converters are introduced and named.

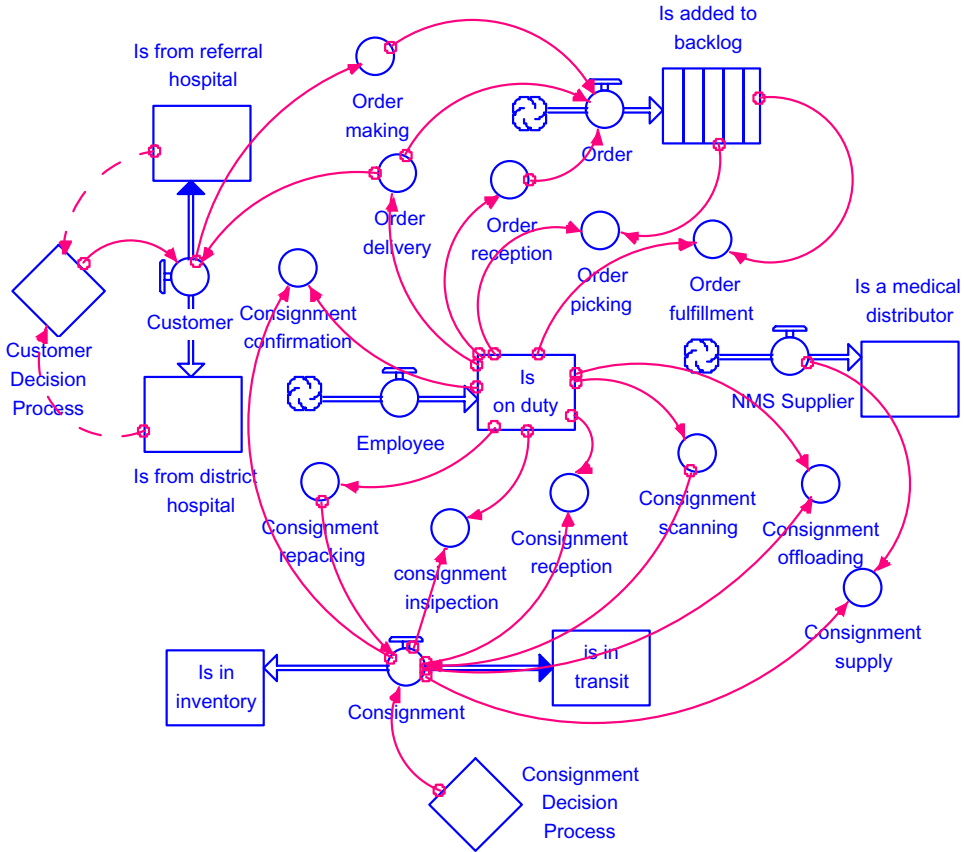


Fig. 21. Information links are identified.

but only extract information out of stocks. To add connectors to the model we follow the ORM constraints, verbalizations plus SD conventions for a stock and flow diagram stated in step 4. In Fig. 21 we show the identified connectors. There are some information connectors in the model, they are from stocks “is from referral hospital”, “is from district hospital”, “is in inventory”, and “is in transit” to DPD. These information connectors are depicted with dashed lines.

Note that most of the connectors in Fig. 21 are from a stock to a converter instead of coming from a flow. This is so because it is the quantities or objects within the flow that influence the variables at the arrow tip of the connectors and not the flow.

Step 5: Create sectors.

As stated in Sec. 5.2, we propose that this step is made optional as practitioners said, it does not affect the model in anyway. Since, having sectors in the model does not necessarily change the model per say but is good when a modeler intends to

carry out sensitive analysis. To avoid repetition of the model in Fig. 21, we do not represent the sectors because there is not much difference.

All in all having applied the ORM to SFD steps to an NMS case, we realize that a model is not complete unless we see and analyze the input variables of the derived model. This is because it is from the input parameters and simulation results that we will be able to further improve the procedure and draw conclusive results. Minus that so far we have been able to see how an underpinned SD model would look like and from that we draw the following evaluations and discussions. Second, the derived SD model shown in Fig. 21 may not have all the required SD elements, e.g. outflows but it gives us a step ahead in underpinning an SD model with a domain modeling method ORM. We therefore believe that if these evaluations are considered, we will not only have a grounded SD model but also a better conceptualization of the SD model plus an improvement in SD model validation hence, reliable SD simulation results.

7. Conclusion

In this paper, we have explored the combination of SD with ORM by identifying relations between ORM and SFD elements, evaluating these relations, defining ORM to SFD transformation steps and applying these steps to a case. The defined steps systematically guide modelers on how to derive an SFD model from an ORM model. However, as one of the practitioners said, these steps can only lead to a “picture” of a stock and flow model. In our further works therefore, we are going to add more steps that cover how to define input parameters in a stock and flow model, what constraints or measure(s) should be taken and show how ORM derivation rules help in defining the stock and flow input parameters.

To evaluate the operationalization of ORM to SFD transformation steps, we used questionnaires and a focus session. This focus group session comprised of participants with knowledge in SD modeling plus other modeling methods. As earlier stated, we follow a design science approach and this approach places additional emphasis on the iterative construction and evaluation of artifacts. We therefore need to conduct more evaluations involving ORM and SD practitioners or experts. These evaluations will help us to improve the procedure and its resulting models.

References

1. J. Forrester, *Industrial Dynamics* (MIT Press, 1961).
2. M. Lebcir, Health Care Management: The contribution of systems thinking, *Business School Working Papers UHBS 2006-7*, <http://hdl.handle.net/2299/683>.
3. G. Richardson, Problems for the future of system dynamics, *Syst. Dyn. Rev.* **12** (1996) 141–157.
4. L. Burmester and G. Matthias, Combining system dynamics and multidimensional modelling — A metamodel based approach, in *Proc. 14th Americas Conf. Information Systems* (Toronto, ON, Canada, 2008).

5. J. Morecroft, A critical review of diagramming tools for conceptualizing feedback system models, *Dynamica* **8**(1) (1982) 20–29.
6. J. D. Sterman, *Business Dynamics — Systems Thinking and Modeling for a Complex World* (McGraw Hill, Higher Education, 2000).
7. A. Golnam, A. van Ackere and A. Wegmann, Integrating system dynamics and enterprise modeling to address dynamic and structural complexities of choice situations, in *Proc. 28th Int. Conf. System Dynamics Society*, ed. T. H. Moon (System Dynamics Society, Seoul, Korea, 2010).
8. K. Saeed, Slicing a complex problem for system dynamics modeling, *Syst. Dyn. Rev.* **8**(3) (1992) 251–261.
9. W. Fey and J. Trimble, An expert system to aid in model conceptualization, in *Proc. 11th Int. Conf. System Dynamics Society* (System Dynamics Society, Cancun, Mexico, 1993), pp. 112–121.
10. D. C. Lane and R. Oliva, The greater whole: Towards a synthesis of system dynamics and soft systems methodology, *J. Oper. Res.* **107**(1) (1998) 214–235.
11. E. Keating, Everything you ever wanted to know about developing a system dynamics model, but were afraid to ask, in *Proc. 16th Int. Conf. System Dynamics Society* (System Dynamics Society, Quebec City, Canada, 1998).
12. T. Halpin, Object-role modeling (ORM/NIAM), *Handbook on Architectures of Information Systems* (Springer, Berlin, Heidelberg, 1998).
13. H. Proper, A. Brecker and S. Hoppenbrouwers, Object-role modeling as a domain modeling approach, in *Proc. Workshop Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD04, held in conjunction with the 16th Conf. Advanced Information systems 2004 (CAiSE 2004))*, eds. J. Grundpenkis and M. Kirikova, Vol. 3 (Riga, Latvia, EU, 2004), pp. 317–328.
14. F. P. Tulinayo, S. Hoppenbrouwers and H. Proper, Integrating system dynamics with object-role modeling, *The Practice of Enterprise Modeling*, eds. J. Stirna and A. Persson, Vol. 15 (Springer, Berlin, Heidelberg, Stockholm, Sweden, 2008), pp. 77–85.
15. P. Chen, The entity-relationship model-towards a unified view data, *ACM Trans. Database Syst.* **1**(1) (1976) 9–36.
16. T. Halpin and G. Wagner, Modeling reactive behavior in ORM, Conceptual Modeling, in *Proc. 22nd ER2003 Conf.*, Vol. 2813 (Springer LNCS, Chicago, 2003), pp. 567–569.
17. F. P. Tulinayo, S. Hoppenbrouwers, P. van Bommel and H. Proper, Integrating system dynamics with object-role modeling and petri nets, in *Enterprise Modelling and Information Systems Architectures*, eds. J. Mendling, S. Renderle-Ma and W. Esswein (GI-Edition, IFIP, Ulm, Germany, 2009), pp. 41–54.
18. F. P. Tulinayo, A. Groessler, S. Hoppenbrouwers and P. van Bommel, Complementing system dynamics with object-role modeling, in *Proc. 27th Int. Conf. System Dynamics Society*, eds. A. Ford, D. Ford and E. Anderson (System Dynamics Society, Albuquerque, New Mexico, USA, 2009).
19. F. P. Tulinayo, P. van Bommel and H. Proper, From a system dynamics causal loop diagram to an object-role model: A stepwise approach, *J. Digit. Inf. Manage.* **10**(3) (2012) 191–203.
20. A. Hevner, S. March, J. Park and S. Ram, Design science in information systems research, *MIS Q.* **28**(1) (2004) 75–105.
21. A. H. M. ter Hofstede, H. A. Proper and Th. P. van der Weide, Formal definition of a conceptual language for the description and manipulation of information models, *Inf. Syst.* **18**(7) (1993) 489–523.
22. T. A. Halpin and M. Curland, Automated verbalization for ORM 2, *On the Move to Meaningful Internet Systems, OTM 2006 Workshops*, Vol. 4278 (Springer LNCS, Heidelberg, 2006), pp. 1181–1190.

23. T. Halpin and T. Morgan, *Information Modeling and Relational Databases*, 2nd edn. (Morgan Kaufmann Publishers, 2008).
24. J. Leaver and C. Unsworth, *System Dynamics Modeling of Spring Behavior in the Orakeikorako Geothermal Field*, Vol. 36, No. 2 (Elsevier Ltd., 2007), pp. 101–114.
25. K. Tan, M. Ahmed and D. Sundaram, Sustainable enterprise modelling and simulation in a warehouse context, *Business Process Management Journal*, Vol. 16 (Emerald Group Publishing Limited, 2010), pp. 871–886.
26. C. A. Heuser, E. M. Peres and G. Richter, Towards a complete conceptual model: Petrinets and entity-relationship diagrams, *Inf. Syst.* **18**(5) (1993) 275–298.
27. T. Halpin and A. Bloesch, Data modeling in UML and ORM: A comparison, *J. Database Manag.* **10**(4) (1999) 4–13.
28. A. Bloesch and T. Halpin, Conceptual queries using ConQuer-II, *Proc. 16th Int. Conf. conceptual modeling*, eds. D. Embley and R. Goldstein, Vol. 1331 (Springer LNCS, Los Angeles, 1997), pp. 113–126.
29. D. W. Stewart, N. P. Shamdasani and D. Rook, *Focus Groups: Theory and Practice*, 2nd edn. (SAGE Publications, Newbury Park, CA, 2007).
30. M. Q. Patton, *Qualitative Evaluation and Research Methods*, 2nd edn. (SAGE Publications, 1990).
31. J. E. Innes, Consensus building: Clarifications for the critics, *Plan. Theory* **3**(1) (2004) 5–20.
32. L. Susskind, S. McKernan and J. Thomas-Larmer, *The Consensus Building Handbook: A Comprehensive Guide to Reaching Agreement* (SAGE Publications, 1999).
33. D. R. Thomas, A general inductive approach for qualitative data analysis, *Population Engl. Ed.* **27**(2) (2003) 237–246.
34. G. Guizzardi, G. Wagner and M. V. Sinderen, An ontologically well-founded profile for uml conceptual models, in *Proc. 16th Int. Conf. Advanced Information Systems Engineering* (CAiSE, Springer, 2004), pp. 112–126.
35. R. Wieringa, W. Jonge and P. Spruit, Roles and dynamic subclasses: A model logic approach, *Proc. European Conf. Object-Oriented Programming* (Springer, 1994), pp. 32–59.
36. T. Halpin, Subtyping revisited, in *Proc. CAiSE'07 Workshops*, eds. B. Pernici and J. Gulla, Vol. 1 (Tapir Academic Press, 2007), pp. 131–141.
37. G. Guizzardi, On the representation of quantities and their parts in conceptual modeling, in *Proc. 2010 Conf. Formal Ontology in Information Systems* (IOS Press, Amsterdam, The Netherlands, 2010), pp. 103–116.