Complex Systems Informatics and Modeling Quarterly (CSIMQ)
eISSN: 2255-9922
Published online by RTU Press, https://csimq-journals.rtu.lv
Article 88, Issue 15, June/July 2018, Pages 45–71
https://doi.org/10.7250/csimq.2018-15.03

Decomposition and Conceptualization to Support System Dynamics Behavior Modeling

Fiona Tulinayo[1]*, Theo van der Weide[2], and Patrick van Bommel[2]

[1]Makerere University, College of Computing and Information Sciences, Uganda
[2]Radboud University, Institute of Computing and Information Sciences, the Netherlands

fturinayo@cit.ac.ug,tvdw@cs.ru.nl,pvb@cs.ru.nl

Abstract. With the increasing need for data-based decision making, social systems and the ecosystems; practitioners and decision makers need guidance in their decision making, as is offered by data-based models and a systematic generation of simulation tools. Overtly, relations between data and practice have been under-conceptualized. Data modelers and decision makers tend to lack a mutual understanding of each other's knowledge systems which has led to huge knowledge gaps. Assimilation of modeling methods therefore is vital. Modeling methods use a specific way of thinking, rules and directions on how to model different aspects of systems. These rules and directions are what we refer to as constructs. Conceptualizing model relations and formulations requires significant domain knowledge and understanding of the constructs. In this article, we use the decomposition mechanism to better conceptualize and understand the System Dynamics (SD) model behavior, and show how using a natural language based domain modeling method (Object-Role Modeling, ORM) helps in dealing with complex SD models. Through applying the decomposition mechanism, we are able to better understand the underlying concepts of the stock and flow diagram and update behaviors of ORM objects. To achieve this, we use examples and an SD model derived from a case "Intrapartum process in Ugandan hospitals" to study the object behaviors. The main results of this article include: a theoretical founding of integrating ORM with SD; quantitative analysis at the level of ORM reasoning; and transformation rules from ORM into SD.

Keywords: System Dynamics, Constructs, Decomposition, Object-Role Modeling.

1 Introduction

Complex systems are characterized by a large number of interacting elements whose overall characteristics cannot be deduced directly from their components. The behaviour of these systems usually is too complex to be modeled by a set of differential equations. Usually, policy makers want to understand these systems sufficiently to develop policies intended to attain optimal system

* Corresponding author

Reference: F. Tulinayo, Th. van der Weide, and P. van Bommel, "Decomposition and Conceptualization to Support System Dynamics Behavior Modeling," Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 15, pp. 45–71, 2018. Available: https://doi.org/10.7250/csimq.2018-15.03

Additional information. Authors ORCID iD: F. Tulinayo – orcid.org/0000-0003-0922-6172, Th. van der Weide – orcid.org/0000-0002-6517-2079. Article PII S225599221800088X. Received: 04 May 2018. Accepted: 11 July 2018. Available online: 31 July 2018.

behaviour. This may be seen as a complex decision problem. Modeling and simulation is increasingly used to provide some understanding that helps policy makers to find such optimal policies.

The analysis of such systems typically requires an interdisciplinary approach combining mathematical and computing science methods with those of environmental, economic, biological and social sciences, often finding a serious gap between the domain's and the ICT languages and methods to bridge.

The increased need for data-based decision making and data integration also requires the combining of various modeling methods [1]. Modeling methods each use a specific way of thinking, rules and directions on how to model an aspect of a system [2]. These rules and directions are what we refer to as constructs. Constructs specify what can be modeled with a given method and define the world view of the method [3]. Here, we use the term constructs as concepts, ideas or images specifically conceived for the purpose of organizing and representing knowledge of interest of a given modeling method [4]. In [5] Wieringa states that understanding the underlying ideas of different methods helps in defining their transition and relations. Yet, understanding relations and formulations requires significant domain knowledge and understanding of the constructs for both the source and target modeling methods. That is to say, the source modeling method constructs are transformed into target modeling method constructs by applying a set of transformation rules. The transformation rules are the smallest entity within a model transformation. They describe how a fragment of the source model can be transformed into a fragment of the target model [6]. Here, we equate fragments to constructs. Putting these fragments together makes a complete set of constructs for a specific modeling method. The transformation rules therefore, help in describing how one or more constructs in the source modeling method can be transformed into one or more constructs in the target modeling method [6]. By understanding the different constructs and transformation rules, constructing a new viewpoint model based on the existing models is realized [7]. For proper model synchronization, it is urged that transformations should consistently propagate changes between different model constructs [8].

The intention of this article is to propose a combination of methods that (1) produces domain descriptions that are understandable by a non-technical domain expert (2) allows for a concise description of complex domains and (3) can be systematically transformed into a simulation tool. In order to be understandable by the domain expert, the method (1a) should be able to handle complexity, and (1b) is communicable at the level of the domain expert.

A major issue during modeling and design is to reduce complexity by introducing various levels of understanding. A most popular method is the decomposition mechanism. This mechanism has been widely applied in different fields, e.g. in method engineering [9], in dynamic mode decomposition (DMD) [10] and in process models [11]. In this article we use the decomposition mechanism to better conceptualize and understand the System Dynamics (SD) model behavior, and show how using a domain modeling method (Object-Role Modeling (ORM)) helps in constructing understandable (and thus valid) SD models.

System Dynamics (SD) is acknowledged as an excellent medium for exploring and identifying knowledge gaps [12]. In SD two diagrams are most commonly used; Causal Loop Diagrams (CLD) and Stock-Flow Diagrams (SFD). CLDs are qualitative in nature while SFDs are quantitative. The qualitative aspect of system dynamics involves the construction of 'influence diagrams' also known as 'Causal Loop Diagrams'. Causal Loop Diagrams are a visual representation of dynamic influences with inter-relationships amongst a collection of SD variables [13]. They are used to brainstorm on a given problem and to qualitatively capture structures and interactions of feedback loops [14], hence, allowing SD modelers understand how changes manifest in a problem domain. The quantitative aspect of system dynamics entails the development of a stock and flow diagram where sets of equations are input into the model resulting into simulations. Graphically, the quantitative aspect displays the relationships between stocks and flows that contain underlying information of the model. Secondly, they offer a basis for rigorous deduction of dynamic behavior because their variables and link distinction can be used to explain a wider range of counter intuitive dynamic phenomena than CLDs [15]. SFDs bring together the modeler's creative thinking ability and their data manipulation ability because they add the dimension of data to mapping of structures which then leads to computer simulation of systems to ascertain the model behavior over time [16]. The derived simulations provide quantitative estimates of system effects and as such, models can be used in a "what if" mode to experiment with alternative configurations, flows and resources [17]. Within the SD community there is consensus on the importance of qualitative data during the development of a system dynamics model, but there is not a clear description about how or when to use it. The lack of a defined systematic procedure on how to obtain and analyze qualitative information creates a gap between the problem modeled and the model of the problem. This causes difficulty in "understanding the links between the observations of reality and the assumptions or formulations in the model especially when the model contains soft variables" [18].

In [19], we noted that SD lacks instruments for discovering and expressing precise language based concepts in domains yet conceptual/domain modeling has long focused on deriving models from natural expressions. In this article therefore, we try to understand the system dynamics relations between observations of reality and formulations through a domain data modeling method, Object-Role Modeling (ORM). We use ORM in particular as an example of a domain modeling language because of its conceptual focus and roots in verbalization, graphical expressiveness and well-defined semantics. 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 the object types 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 roles. 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) [20] and ORM2 [21] are examples of such generic systems for constructing sentences. These sentences will be referred to as information descriptors. The notion of information descriptors was introduced under LISA-D (Language for Information Structure and Access Descriptions) which is based on PSM (Predicator Set Model) in [20]. With respect to decomposition, the data modeling technique PSM [20] introduces the *schema type* as a mechanism for decomposition. Besides PSM introduces the grammar type for semi-structured data, allowing the PSM schema to be extended with a grammatical description (which can be compared to a DTD in XML). However, PSM does not cater for the behavioral description of objects. In [22] abstraction layers for data modeling are introduced at a more fundamental level. Several additional methods have been proposed to combine data modeling with behavioral descriptions, such as state charts (see for example [23]). UML (see for example [24]) offers modeling techniques for many aspects of software development, such as a class model and behavior description. PSM^2 ([25]) is an action-based approach to model an application domain, starting from a sample behavioral description (called a logbook, see [26]). PSM^2 does not have a mechanism for decomposition in the behavioral description of object types. For similar applications, see [27], [28] and [29].

The methodology we use is based on the Design Science Research Method (DSRM) which is the standard research methodology used in the Information Systems discipline for designing new artifacts that solve unsolved problems or improve upon existing solutions (see [4], [30]). Based on a broad design science literature review, inspired by the seminal paper of [4], [31] proposes a methodology for design science research consisting of the following six steps: 1. Problem identification and motivation: Define the specific research problem and justify the value of a solution. 2. Definition of the objectives for a solution: Infer the objectives of a solution from the problem definition and knowledge of what is possible and feasible. 3. Design and development: Create the artifact. Such artifacts are potentially constructs, models, methods, or instantiations. 4. Demonstration: Demonstrate the use of the artifact to solve one or more instances of the problem. This could involve its use in experimentation, simulation, case study, proof, or other appropriate activity. 5. Evaluation: Observe and measure how well the artifact supports a solution to the problem. Depending on the nature of the problem venue and the artifact, evaluation could take many forms. 6. Communication: Communicate the problem and its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness to researchers and other relevant audiences, such as practicing professionals, when appropriate. Our work is organized in accordance with these 6 steps as follows. Problem identification and motivation and the definition of the objectives for a solution are described in the introduction of this article. Design and development are the topic of Sections 3 and 4. Sections 5 and 6 provide a demonstration of the proposed method. The conclusion of this article addresses the evaluation of the method proposed in this article. The final step, communication, can be found throughout the article.

The rest of this article proceeds as follows. In Section 2, we present the basic concepts and constructs of the domain modeling method under consideration (ORM). We use the decomposition mechanism of this method to define the SD method. Therefore, we present SD constructs with their underlying principles in Section 3. In order to achieve a solid theoretical founding of integrating ORM with SD, we start by presenting causal loop diagram constructs and their underlying principles, along with formal definitions of causal loop diagrams in Subsection 3.1. Next we consider stock and flow diagrams in Subsection 3.2. The corresponding quantitative analysis at the level of ORM reasoning is presented in Section 4. Our approach has been applied in a real life context, which is described in Section 5. Here, we use the case study Intrapartum process in Ugandan Hospitals to demonstrate that these ORM and SD concepts facilitate ORM to work as a foundation for SD. After constructing the model, it is decomposed by first separating the object types and then treating each object type independently. This leads to the definition of all influencing relations. In Section 6 the design from Section 5 is realized with a concrete SD tool. In Section 7 we shortly evaluate the merits of our proposed method, draw some conclusions, and suggest some further research, directions.

2 ORM Concepts and Constructs

ORM's basic building blocks include: entity types (object types), value types and roles [32][3]. 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. Entity types are designated by solid-line named ellipses in the graphical reproduction of the information grammar. All entity types have a reference scheme, which may be simple (either a reference mode, or an entity to entity relationship) or compound. These reference schemes indicate how a single value relates to that entity 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. 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. To represent some of these definitions let us use an example of the procedures a patient might go through en route from arrival to discharge. The procedures are stated as:

S1 Patient (Name) arrives at Hospital (Name) at Time (.am/p.m.).
S2 Patient (Name) queues up.

[3] For ORM terminologies in this study, we used Halpin and Morgan [32]; and to model ORM models we used Natural ORM Architect (NORMA).

S3 MedicalPerson (Name) examines Patient (Name) to establish Patientillness (Name).
S4 Patient (Name) is treated by MedicalPerson (Name).
S5 Patient (Name) is discharged by MedicalPerson (Name).

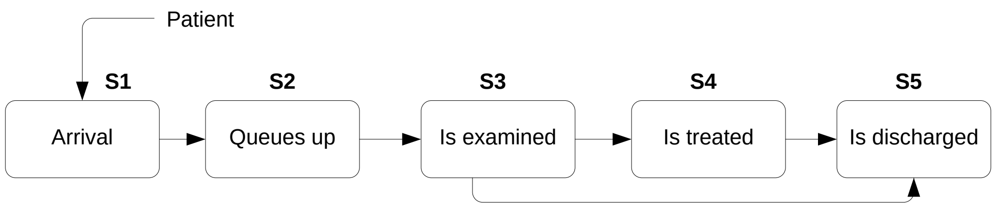


Figure 1. Various Patient states

The procedure presented in Figure 1 basically describes the various subsequent states recognized for the object type Patient. In this article, we further explore the notion of state presented in Figure 1, and how states come in naturally during the modeling process. The conceptual structure of this example is represented on an ORM diagram in Figure 2.

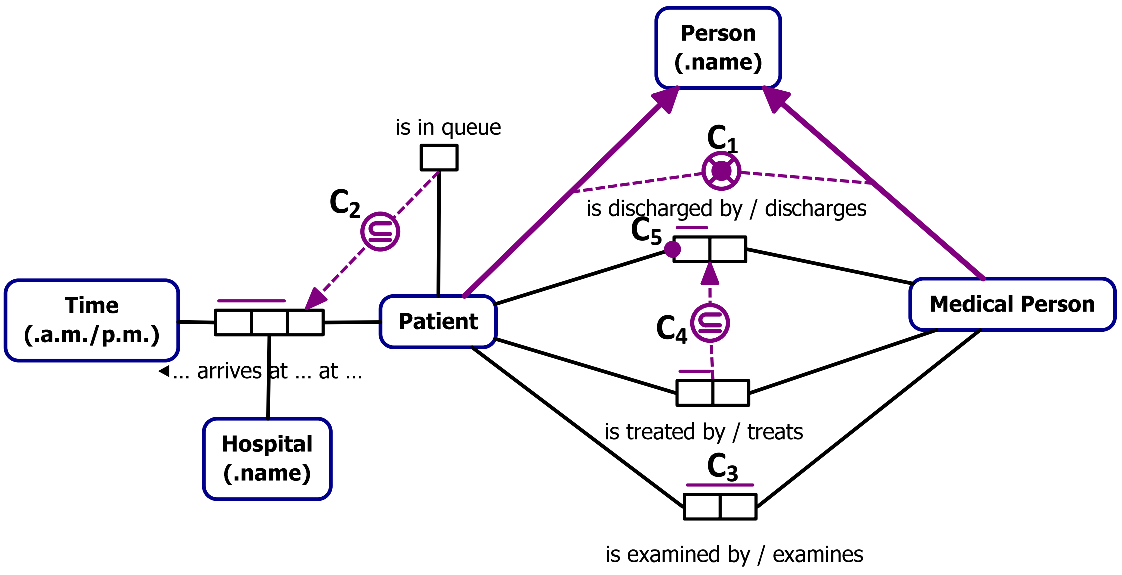


Figure 2. ORM Patient flow concepts

C1 (*Exclusive or*): **For each** Person, **exactly one of the following holds:**
some Patient is **that** Person; **some** Medical Person is **that** Person.
C2 (*Subset*) **If some** Patient is in queue **then that** Patient arrives at **some** Hospital at **some** Time.
C3 Medical Person examines Patient. **It is possible that more than one** Medical Person examines **the same** Patient **and that more than one** Patient is examined by **the same** Medical Person.
Each Patient is treated by at most one Medical Person.
C4 **If some** Medical Person treats **some** Patient **then that** Medical Person discharges **that** Patient.
C5 (*mandatory*): ***each*** Patient is discharged by ***at least one*** Medical Person.

Further, we demonstrate these concepts in a case “Intrapartum process in Ugandan Hospitals”.

3 SD Constructs and Underlying Principles

As already stated, the SD notations (CLDs and SFDs), each have a number of constructs with underling principles[4]. In subsections 3.1 and 3.2 respectively, we present their basic building blocks and underlying principles.

3.1 Constructing a Causal Loop Diagram

A Causal Loop Diagram is made up of *variables*, *signs* (either a positive or negative) and *causal links* with arrows representing the causal influence. The arrows are drawn in a circular manner indicating the cause and effect leading to a feedback loop which is a closed sequence of causes and effects sometimes referred to as a closed path of action and information [33]. When constructing a CLD, there are design rules to be followed.

Design Rule 1 (Positive sign): A causal link from one element '*A*' to element '*B*' is positive $(+)$ if either '*A*' adds to '*B*' or a change in '*A*' makes variable '*B*' change in the *same* direction.

Design Rule 2 (Negative sign): A causal link from one element '*A*' to another element '*B*' is *negative* (-) if either '*A*' subtracts from '*B*' or a change in '*A*' makes '*B*' change in the *opposite* direction. In addition to the signs on each link, is a complete loop sign (either a positive (Reinforcing) or negative (Balancing). The sign for a particular loop is determined by counting the number of minus (-) signs on all the links that make up that loop.

Design Rule 3 (Reinforcing Loop): A feedback loop is called *positive or reinforcing*, indicated by a plus or *'R'* sign in parentheses, if it contains an even number of negative causal links.

Design Rule 4 (Balancing Loop): A feedback loop is called *negative or balancing*, indicated by a minus or *'B'* sign in parentheses, if it contains an odd number of negative causal links.

Thus, the sign of a loop is the algebraic product of the signs of its links. Often a small looping arrow is drawn around the feedback loop sign to more clearly indicate that the sign refers to the loop (*see Rule 3 and 4 in figures presented in Figure 3*). Further explanation on how CLD influences operate can be found in [12].

Design Rule 5 (Delay Mark/Time Delay): Between variables *'B'* and *'A'* in Rule 5, Figure 3, is a delay mark. This delay mark implies that there is a time lapse (lag) between these variables before the actual change is noticed or becomes evident. Delays are of two types: material delays and information delays. Material delays represent a lag in the physical flow while information delays represent gradual adjustment of people's beliefs. Identifying delays is an important step in the system dynamics modeling process because they often alter a system's behavior in significant ways. The longer the delay between cause and effect, the more likely it is that a decision maker will not perceive a connection between the two [12]. A detailed explanation of delays can be found in [12] p. 409.

3.1.1 Formal Definition of Causal Loop Diagram

As stated earlier, a Causal Loop Diagram is made up of *variables* and *causal links* with arrows representing the causal influence. Causal links have associated a *sign* (either a positive or negative) and may have an associated *delay*. A causal link expresses a causal relationship between two factors. If the link has an associated positive sign, then the link expresses a positive influence/relation. We write $F \curvearrowright^{+} G$ (see Design Rule 1) to express that a change in variable F causes a similar change in variable G. We assume there is a time delay between the cause and its effect; this time delay does not have a lowerbound on its duration (which makes it different from the delay that may be associated with a causal link). When the causal link is effected at time t, then it relates the situation of the cause at time t^- with the effect at time t^+ (using standard notation for calculus [34]).

[4] For SD terminologies used in this study we use Sterman [12]. All SD stock and flow diagrams are drawn using an SD software called STELLA and all Causal Loop Diagrams were drawn using Vensim. This is because the two platforms are easy to use and offer a practical way to dynamically visualize and communicate how complex systems and ideas really work.

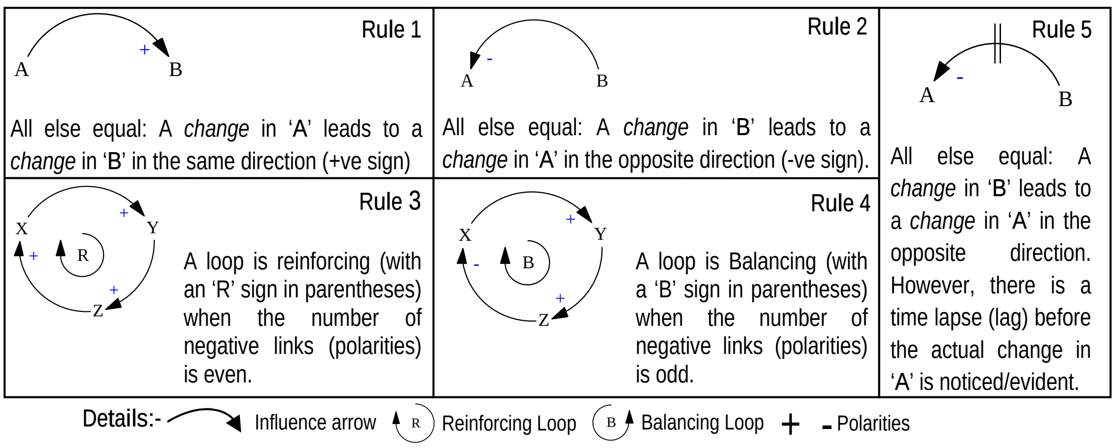

Rule 1
A
+
B
All else equal: A *change* in 'A' leads to a *change* in 'B' in the same direction (+ve sign)
Rule 2
A
-
B
All else equal: A *change* in 'B' leads to a *change* in 'A' in the opposite direction (-ve sign).
Rule 5
A
-
B
All else equal: A *change* in 'B' leads to a *change* in 'A' in the opposite direction. However, there is a time lapse (lag) before the actual change in 'A' is noticed/evident.
Rule 3
X
+
Y
R
+
+
Z
A loop is reinforcing (with an 'R' sign in parentheses) when the number of negative links (polarities) is even.
Rule 4
X
+
Y
B
-
+
Z
A loop is Balancing (with a 'B' sign in parentheses) when the number of negative links (polarities) is odd.
Details:-
Influence arrow
R
Reinforcing Loop
B
Balancing Loop
+
-
Polarities

Figure 3. A summary of the causal loop diagram rules

Lemma 1. *$\curvearrowright^{+}$ is a transitive relation.*

Proof. Suppose $F \curvearrowright^{+} G$ and $G \curvearrowright^{+} H$. Then, because of $F \curvearrowright^{+} G$, a change of variable F causes a similar change to variable G, which in turn leads to a similar change of variable H hence $G \curvearrowright^{+} H$. Consequently, a change in variable F leads to a similar change of variable H, or, $F \curvearrowright^{+} H$. ◇

A causal link that has associated a minus-sign (see Design Rule 2) expresses a negative influence relation. So the sign of change in the effect variable is opposite to the change in the cause variable. We use the notation $F \curvearrowright^{-} G$ for this case. However, the relation $\curvearrowright^{-}$ is not a transitive relation:

Lemma 2. *If $F \curvearrowright^{-} G$ and $G \curvearrowright^{-} H$, then $F \curvearrowright^{+} H$.*

Proof. Suppose $F \curvearrowright^{-} G$ and $G \curvearrowright^{-} H$. Then, because of $F \curvearrowright^{-} G$, a change in variable F causes a change in variable G in the opposite direction. This change of variable G, because of $G \curvearrowright^{-} H$, leads to a change in variable H in the direction opposite to the change in variable G. Consequently, a change of variable F leads to a similar change of variable H, or $F \curvearrowright^{+} H$. ◇

Next we consider the combination of positive and negative influence.

Lemma 3.
If $F \curvearrowright^{+} G$ and $G \curvearrowright^{-} H$, then $F \curvearrowright^{-} H$.
If $F \curvearrowright^{-} G$ and $G \curvearrowright^{+} H$, then $F \curvearrowright^{-} H$.

Proof.
Suppose $F \curvearrowright^{+} G$ and $G \curvearrowright^{-} H$. Then, because of $F \curvearrowright^{+} G$, a change of variable F causes a change of variable G in the same direction. This change of variable G, because of $G \curvearrowright^{-} H$, leads to change of variable H in the direction opposite to the change in variable G. Consequently, a change in variable F leads to an opposite change of variable H, or, $F \curvearrowright^{-} H$.

Suppose $F \curvearrowright^{-} G$ and $G \curvearrowright^{+} H$. Then, because of $F \curvearrowright^{-} G$, a change of variable F causes a change in variable G in the opposite direction. This change of variable G, because of $G \curvearrowright^{+} H$, leads to change of variable H in the same direction as the change of variable G. Consequently, a change in variable F leads to an opposite change of variable H, or, $F \curvearrowright^{-} H$. ◇

Let $F_1,\ldots,F_n$ be variables, such that $F_i \curvearrowright^{+} F_{i+1}$ or $F_i \curvearrowright^{-} F_{i+1}$ for each $i \leq 1 < n$, then we call $[F_1,\ldots,F_n]$ a causal path from F_1 to F_n. This brings us to the following conclusion:

Lemma 4. *Let P be a causal path from F to G, then we have:*
$F \curvearrowright^{+} G$ if the number of negative influences in path P is even
$F \curvearrowright^{-} G$ if the number of negative influences in path P is odd

Proof. We prove the statement by induction on the length of the causal path P.

- For a length 1 path $P = [F, G]$, we have the following cases: (1) 1. $F \curvearrowright^{+} G$: then also the number of negative influences on path P is even; 2. $F \curvearrowright^{-} G$: then also the number of negative influences on path P is odd.
- Suppose the property holds for paths of length n, let P be a path of length $n+1$ from F to H. Then we can decompose P as a path of length n from F to some G, and path of length 1 from G to H. We have the following cases: (1) 1. $G \curvearrowright^{+} H$: then the number of negative influences on path $[F, G]$ is the same as on the path $[F, H]$. The property now follows from Lemma 1, Lemma 3 and the induction hypothesis. 2. $G \curvearrowright^{-} H$: this case is similar to the previous case.

◇

A path P from F to F is referred to as a causal loop. The following lemma formalizes Design Principles 3 and 4:

Corollary 1. *If loop P from F to F contains an even number of negative influences, then we can conclude $F \curvearrowright^{+} F$. This means that any change of variable F is reinforced by loop P. On the other hand, if path P contains an odd number of negative influences, then any change of variable F is damped by an opposite change by loop P.*

3.2 Stock-Flow Diagram Constructs and Underlying Principles

The Causal Loop Diagram describes variables and how they influence each other. The Stock-Flow Diagram is a materialization of the Causal Loop Diagram, as an easy to use framework for setting up differential equations. The Stock-Flow Diagram is made up of the following building blocks: *stocks*, *flows* (inflow and outflows), *converters* (auxiliary and constant), *sources* and *sinks*.

3.2.1 Constructing a Stock-Flow Diagram

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 bi-flows as represented in Figure 4. An uniflow means that information in that flow moves (flows) in one direction only and the flow takes on non-negative values only. A bi-flow on the other hand, can take on any value and information flows in two directions. Flows originate from a *sources* and terminate in a *sink* which are depicted as clouds.

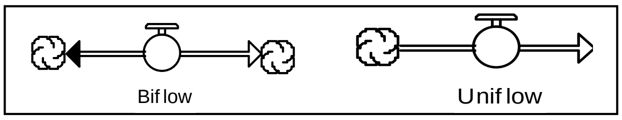


Figure 4. Types of flow

Stocks are depicted as boxes and are defined as containers (reservoirs) containing quantities describing the state of the system. The value of stocks changes overtime through flows (inflows and outflows) [35].

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 form of equations or values that can be applied to stocks, flows, and other converters in the model [36]. *Constants* are state variables which do not change [35]. 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 [37]. 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 [36]. Lastly, we have the concept of *sectors* which are subsystems or subcomponents within a system. They hold/handle all decisions, stocks, and information about a particular element or area; and contain different information used in an information system.

Note that among converters we only mention auxiliary and constants but not exogenous variables as building blocks. This is because for exogenous variables, although they are part of the SFD model, their values are determined by factors outside the model. Secondly, 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 SFD building blocks except sectors in Figure 5 followed by some of the SFD design rules.

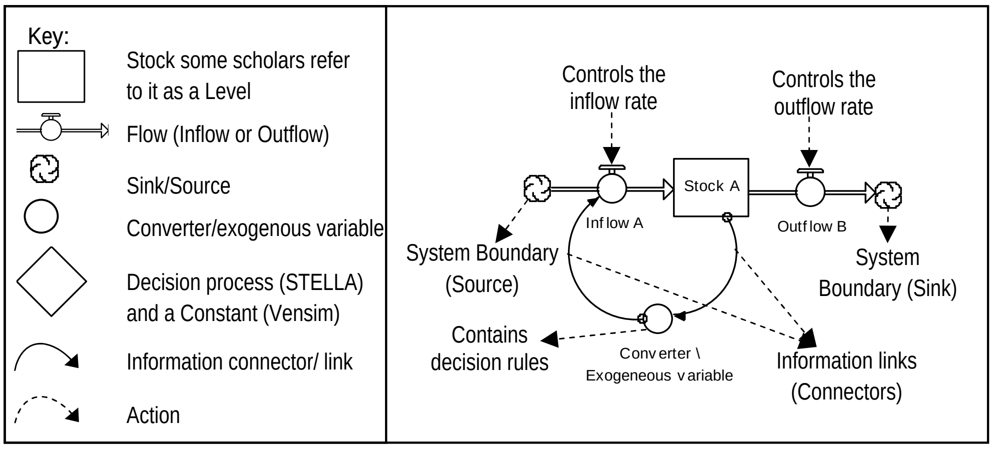

Key:
Stock some scholars refer to it as a Level
Flow (Inflow or Outflow)
Sink/Source
Converter/exogenous variable
Decision process (STELLA) and a Constant (Vensim)
Information connector/ link
Action
Controls the inflow rate
Controls the outflow rate
Stock A
Inflow A
Outflow B
System Boundary (Source)
System Boundary (Sink)
Contains decision rules
Converter \ Exogeneous variable
Information links (Connectors)

Figure 5. A summary of SFD basic building block (Source: [38])

Design Rule 6 (Flows): In system dynamics, every flow is influenced by another variable (stock or converter) in the model through connectors (information links). This enables the 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. Thus flows can be influenced by stocks, converters and exogenous variables, but cannot directly influence converters and exogenous variables or other flows.

Design Rule 7 (Converters): As we stated earlier there are two types of converters: a constant and an auxiliary. *Converters are 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 8 (Sink and Source): A sink and source exist on flows that do not originate from or terminate into a stock.

Design Rule 9 (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.

Design Rule 10 (Stocks): A stock is the visualization of the variable of the Causal Loop Diagram. Typically variables represent the relevant states that are distinguished for that type of object. A stock represents a state. A state change corresponds to an object flowing from one stock into another. Consequently, an object can be in at most one stock at each moment. Stocks are directly influenced by inflows and indirectly by information links. 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. The size of stock S at time t is denoted as $Stock_t(S)$ thus defining stock as a basic concept.

We will give a property of stock in terms of differential equations below. Let $Flow_t(S \to T)$ be the size of the flow from stock S to stock T via the flow from S to T. The total inflow in stock S via any incoming flow at time t is denoted as $Inflow_t(S)$, the total, outflow is denoted as $Outflow_t(S)$ (see [12]). So we have:

$$Inflow_t(S) \triangleq \sum_{T \to S} Flow_t(T \to S) \tag{1}$$

$$Outflow_t(S) \triangleq \sum_{S \to T} Flow_t(S \to T) \tag{2}$$

The stock accumulation is expressed as: at any moment $t \geq t_0$. Where t_0 is the starting time.

$$Stock_t(S) = Stock_{t_0}(S) + \int_{t_0}^{t} [Inflow_u(S) - Outflow_u(S)]\, du \tag{3}$$

As a consequence the change of flow is expressed as:

$$\frac{d(Stock_t(S))}{dt} = Inflow_t(S) - Outflow_t(S) \tag{4}$$

4 The Formal Approach

The approach in this study is based on conventional fact-based ORM but extended to cover some of the aspects of PSM[2]. We assume that each state of an object is derivable from its (modeled) properties. As a consequence, each state can be described by some information descriptor. A base for an object type describes a set of possible states for that object type. Note that an information descriptor [20] can be seen as a path through a conceptual schema, describing a relation between its starting and its ending object types. We restrict ourselves to homogeneous information descriptors (see [20]), meaning that the descriptor has both a unique starting point and a unique ending object type. This is explained as follows: Let $Start(D)$ be the set of starting points of information descriptor D and $End(D)$ its set of ending points. Furthermore, we use $Top(X)$ to denote the pater familias of the object type hierarchy to which object type X belongs (see [20]). We call an information descriptor D homogeneous if (1) all object types in $Start(D)$ have the same pater familias (referred to as $Top(Start(D))$), and (2) all object types in $End(D)$ also have the same pater familias (referred to as $Top(End(D))$). Then the evaluation of D at point of time t leads to the pairs of object instances (x,y) such that $x \in Pop_t(Top(Start(D)))$ and $y \in Pop_t(Top(End(D)))$ that are related via the D (see [39]).

A set of information descriptors is known as a conceptual base, where each information descriptor of a conceptual base describes a typical state of its starting object type. We will call $\mathcal{D}$ a base for object type X if all descriptors in $\mathcal{D}$ start from object type X:

$$\forall_{D \in \mathcal{D}} [Top(Start(D)) = X]$$

Base $\mathcal{D}$ is called complete for object type X if at each point of time t:

$$\bigcup_{D \in \mathcal{D}} (Pop_t(D)) = Pop_t(X)$$

For a detailed discussion see [40].

The current state of the Universe of Discourse is recorded by the corresponding ORM scheme as the population of this scheme with all valid (elementary) facts in that particular state at that moment. Consequently, each information descriptor will have a well-defined result. Let $Pop_t(D)$ be the result of information descriptor D at time t (see also [39]). The goal of applying System Dynamics on an ORM scheme is to obtain quantitative insight.

The quantitative insight may be on the complete ORM scheme, or we may want to focus on a particular part of the scheme. Depending on our needs, we will choose a set $D_1, \ldots.., D_n$ of information descriptors that correspond to relevant aspects. Typically, an information descriptor will refer to instances of an object type in a particular state. We then will be interested in the amount and growth behavior of such objects, expressed as $P_t(D_i)$ for descriptor D_i $(1 \leq i \leq n)$, using $P_t(D_i)$ as a shorthand for the number of instances $\|Pop_t(D_i)\|$ in the population of D_i at time t. In terms of System Dynamics, these information descriptors are the factors or variables.

For an overall description of the quantitative behavior of an ORM scheme Σ, the goal is to find a set of factors $\{D_1, \ldots.., D_n\}$ of information descriptors that is complete for Σ, meaning:

C-1 The variables are independent: $Pop_t(D_i) \cap Pop_t(D_j) \Rightarrow i = j$

C-2 From $Pop_t(D_1), \ldots, Pop_t(D_n)$ we can derive $Pop_t(X)$ for each object type X.

C-3 We can describe the quantitative behavior of $D_1, \ldots, D_n$ by a system of differential equations in terms of $P_t(D_1), \ldots, P_t(D_n)$.

If $\mathcal{D}$ is a complete set of information descriptors for Σ, then $D \in \mathcal{D}$ is superfluous if $\mathcal{D} - \{D\}$ also is a complete set for Σ. A set $\mathcal{D}$ of factors that is complete for Σ, is called a base for that scheme if this set $\mathcal{D}$ does not contain superfluous information descriptors. In that case we refer to the factors as dimensions of the scheme. The dimension of an ORM scheme Σ is the minimal number of dimensions required for a base of this scheme. We call $D_1, \ldots, D_n$ a conceptual base for scheme Σ if it satisfies property C-1. We call a conceptual base a computational base for scheme Σ if it also satisfies property C-3. A complete base for scheme Σ is a computational base that also satisfies property C-2.

Let D and E be information descriptors, then there is a flow from D into E, denoted as $D \rightarrow^+ E$ if instances from D may move to E. More precisely:

$$D \rightarrow^+ E \;\triangleq\; \exists_{Pop,x,s<t} \left[x \in Pop_s(D) \wedge x \in Pop_t(E) \right] \tag{5}$$

meaning that in some population Pop there is an instance x from D that on a later moment is an instance of E. Then we define $\rightarrow$ as the one-step subrelation of $\rightarrow^+$. We will refer to the relation $\rightarrow$ as the flow relation. A base for a scheme Σ with its induced flow relation $\rightarrow$ form the base for the SD simulation of Σ. Next we motivate that an SD indeed can simulate an ORM scheme.

Let $\{D_1, \ldots, D_n\}$ be a computational base for scheme Σ and $\rightarrow$ the induced flow relation. Since the variables $P_t(D_1), \ldots, P_t(D_n)$ (as functions of t) take discrete values, it is not obvious that differential equations can be used to describe their behavior. In System Dynamics, rather than determining these differential equations, causal influences between the variables (factors) are detected, leading to a Causal Loop Diagram. That way we can detect basic system properties such as enforcing loops. Another opportunity we have is that we can derive a differential scheme describing the flows between the variables such that we can simulate system behavior in a stepwise manner, leading to the Stock-Flow Diagram. Here we focus on such differential schemes. In this subsection we describe the formal relation between ORM schemes and differential schemes.

Assume we have successive time steps $t_0, \ldots, t_n$, with $t_{i+1} = t_i + \Delta t$. Then the population size of variable D at time t_n is the cumulation of the changes $dP_{t_i}(D)$ during the intervals $[t_i, t_{i+1}]$:

$$P_{t_n}(D) = P_{t_0}(D) + \sum_{i=1}^{n} dP_{t_i}(D) \tag{6}$$

During the period $[t_{i-1}, t_i]$ the change may also be described as (using Formula 7):

$$dP_{t_i}(D) = \sum_{E \rightarrow D} \left(\int_{t_{i-1}}^{t_i} Flow_s(E \rightarrow D)ds \right) - \sum_{D \rightarrow E} \left(\int_{t_{i-1}}^{t_i} Flow_s(D \rightarrow E)ds \right) \tag{7}$$

Flows are approximated as follows:

$$\int_{t}^{t+\Delta t} Flow_s(D_1 \rightarrow D_2)ds \;\approx\; Rate_t(D_1 \Rightarrow D_2) \cdot P_t(D_1) \cdot \Delta t \tag{8}$$

where $Rate_t(D_1 \Rightarrow D_2)$ is the fraction of the objects in D_1 that flow from D_1 to D_2 per unit of time, at time t. So we have:

$$Rate_t(D_1 \Rightarrow D_2) = \lim_{\Delta t \to 0} \frac{\int_t^{t+\Delta t} Flow_s(D_1 \to D_2) ds}{\Delta t} / P_t(D_1) \tag{9}$$

Suppose information descriptors D_1 and D_2 describe two different states of some object type X, such that instances may flow from state D_1 to state D_2. Note that, according to **C-2**, at any moment $Pop_t(D_1)$ and $Pop_t(D_2)$ are disjoint. The instances of $Pop_{t+\Delta t}(D_2) \cap Pop_t(D_1)$ may be assumed to have flown from D_1 into D_2 between t and $t + \Delta t$, provided Δt is sufficiently small. Consequently, we have proved that the flow may be expressed as a rate of the source stock as follows:

Theorem 1.

$$Rate_t(D_1 \Rightarrow D_2) = \lim_{\Delta t \to 0} \frac{\|Pop_{t+\Delta t}(D_2) \cap Pop_t(D_1)\| \,/\, \|Pop_t(D_1)\|}{\Delta t} / P_t(D_1) \tag{10}$$

In general the rate is not easily measured. However, in the case of a simulation, the error introduced by an incorrect rate estimate, may have a limited effect only. In SD applications, the rate associated with each link is either taken as a constant fraction, or, in case of a converter, as a parameterized fraction. Note that the proof of this theorem is the explanation above.

4.1 A Continuous Approximation

Sofar we have extended ORM with the concepts of *state* and *state transition* [40]. However, extending ORM with state transitions is not new. In [41], [42] they explore the extension of ORM to support declarative specification of dynamic rules restricted to single-step transitions. The extension of ORM in [40] allowed us analyze SD model behavior at a conceptual level and provide an inductive definition by presenting a mechanism to use the information structure for describing information structure states and their relations in time.

For a deeper understanding of the dynamics of the conceptual system described by the ORM model we focus on groups of objects rather than individual object instances and their transitions. Typically, we focus on (compound) states and their size, and how these sizes vary over time.

Our intention is to analyze the dynamics by continuous simulation. In [43], Albrecht defines continuous simulation as a computer model of a physical system that continuously tracks response of a system over time according to a set of defined equations typically involving differential equations. More concretely, Lee [44] states that a computer simulation or computer model, attempts to create and analyze an abstract model or program that simulates the behaviors of real-world systems.

To facilitate the analysis, and to find the differential equations involved, we apply a continuous approximation of the discrete world. We assume time to be continuous, thus taking values from the real numbers ($\Re$). According to [45], the *closed-world transformation* is the most popular continuous-to-discrete transformations in digital system design and is also used in digital simulation. Typically for a System Dynamics is to use a fixed sample interval. At this point we have two options:

1. Assuming that population sizes also take values from the continuous domain $\Re$. Then differential equations can be used to describe the system behavior. Differential equations are a powerful mechanism to derive properties of a system. From a differential equation a differential scheme is easily derived.
2. Setting up a system of differential equations directly.

We feel that a system of differential equations more adequately can describe system behavior. In this subsection we discuss the basis and motivation for this approximation in a formal way. In the next subsection we introduce simulation as an effective realization for this formal approach. Basically we show how a System Dynamics interpretation can be done at conceptual level.

4.1.1 Influencing Transitions

In Section 3.1, causal relations were introduced as $D_1 \curvearrowright^{+} D_2$ or $D_1 \curvearrowright^{-} D_2$ between (compound) states of objects, expressing a positive or negative effect of a change in the population of D_1 at time t on the population of D_2 at time t. We assume there is a time delay between the cause and its effects; this time delay does not have a lowerbound on its duration, and may be seen as infinitely small. When the causal link is effected at time t, then it relates the situation of the cause at time t^- with the effect at time t^+ (using standard notation for calculus [34]). The causal relations are defined as follows:

$$D_1 \curvearrowright^{\mathbf{op}} D_2 \triangleq \mathsf{Sign}\left(\frac{d\,Pop_t(D_1)}{dt}\right)(t^-) = \mathbf{op}\ \mathsf{Sign}\left(\frac{d\,Pop_t(D_2)}{dt}\right)(t^+) \ \textit{at each moment } t \tag{11}$$

for $\mathbf{op} \in \{-,+\}$. As an example, considering Figure 13 an increase of incoming patients leads to an increase in the number of patients in queue. This is expressed as:

$$\textit{Patient arrival} \curvearrowright^{+} \textit{Patients in queue}$$

Another rule may be that a decline in the available medical persons is to be followed by an increase in the number of patients in queue:

$$\textit{Available medical persons} \curvearrowright^{-} \textit{Patients in queue}$$

When more beds are being used for delivery, then there are less admission beds, and vice versa:

$$\textit{Admission Bed} \curvearrowright^{-} \textit{Delivery Bed}$$
$$\textit{Delivery Bed} \curvearrowright^{-} \textit{Admission Bed}$$

Besides rules set up by the system analyst, the relations $\dashrightarrow$ and $\curvearrowright^{\mathbf{op}}$ also are related in the following way:

1. If $D_1 \dashrightarrow D_2$ then also $D_1 \curvearrowright^{+} D_2$.
2. If both $D_1 \dashrightarrow D_2$ and $D_1 \dashrightarrow D_2$ then also $D_2 \curvearrowright^{-} D_3$ and $D_3 \curvearrowright^{-} D_2$.

Causal influences are special kinds of growth relations between states of object types. We call the causal relation $D_1 \curvearrowright^{\mathbf{op}} D_2$ homogeneous when also $D_1 \dashrightarrow D_2$. In the other case, the causal relation is called a converter. For instance, an increase in the number of patients will lead to an increase of the number of beds:

$$\textit{Patient} \curvearrowright^{+} \textit{Bed}$$

This is an example of a converter. The statement expresses the fact that there will be more new beds as a result of more patients in the hospital. So the number of patients positively influences the transition $\omega \dashrightarrow \textit{Bed}$.

$$\textit{Depends}(\textit{Patient}, \omega \dashrightarrow \textit{Bed})$$

We call a (compound) state x a *start state* of compound state y if x is an initial state and also a direct substate of y:

$$\textit{StartState}(x,y) \triangleq x \in \mathcal{D}_{\mathsf{in}} \land \textit{SubState}_1(x,y)$$

We call x a birth state if it is an initial state but not the start state of another state:

$$\textit{BirthState}(x) \triangleq x \in \mathcal{D}_{\mathsf{in}} \land \neg \exists_y [\textit{SubState}_1(x,y)]$$

If x is a birth state, then we also write:

$$\omega \dashrightarrow x$$

where ω is virtual (source) state. If x is a death state, then we also write:

$$x \dashrightarrow \Omega$$

where Ω also is a virtual (sink) state. Analogously we introduce final state and death state:

$$\begin{aligned} \mathit{FinalState}(x,y) &\triangleq x \in \mathcal{D}_{\mathsf{out}} \wedge \mathit{SubState}_1(x,y) \\ \mathit{DeathState}(x) &\triangleq x \in \mathcal{D}_{\mathsf{out}} \wedge \neg \exists_y \left[\mathit{SubState}_1(x,y)\right] \end{aligned}$$

We call the structure $\mathcal{B}(X) = \langle \mathcal{S}, \mathit{SubState}_1, \mathcal{D}, \dashrightarrow, \mathcal{D}_{\mathsf{in}}, \mathcal{D}_{\mathsf{out}} \rangle$ a behavioral description of object type X. Note that an object type may have more than one behavioral description, but this will be generally not the case in practice. Further discussion on states (decomposition and transition) is presented in [40].

4.2 Towards Operationalizing the Process

Complex system dynamics models are hard to conceptualize because modelers or stakeholders cannot understand how various parts of the system interact and add up to the whole [46]. To facilitate operationalization of SD with ORM underpinning process, the following guidelines were identified as a means to support the SD-ORM transformation. (1.) *Develop a CLD model*: The first step when developing a system dynamics model is normally constructing a CLD. This CLD model represents articulated mental models. The CLD modeling process, however, does not impose many restrictions and does not separate structure from behavior. But it helps to express and organize knowledge and assess learning about complex situations [47], [48]. CLDs are fundamental at articulating and understanding how variables influence each other. (2.) *Transform the CLD model into an ORM model*: This step is important because it helps clearing existing ambiguities in the CLD model; and, it improves SD model conceptualization (refinement and specification of abstract concepts). A detailed explanation of this step is presented in [49]. (3.) *Decompose the ORM model into schemes*: Decomposition is the disintegration or breaking down of a given ORM model into small manageable fragments. The decomposition guiding steps include:

1. *Separate object types with their roles into unique ORM schemes:* In most cases, object types in an ORM model are more than one and contain different objects. To apply the decomposition mechanism to an ORM model, object types should be separated into different ORM schemes. However, in some cases the ORM model has supertypes and for each supertype there is a hierarchy of substates. It is therefore important to separate these object types so that the modeler is able to represent states for each ORM schema with a unique identifier.
2. *Handle each object type independently:* In order to improve conceptualization of the SFD model, handle each object type independently. Doing so enables a modeler understand how objects in each state relate to one another. For each object type;
   a) Give each role a unique identification.
   b) Categorize states into elementary and compound states.
   c) Represent states in order of activity occurrence.
   d) Add directed paths.

   Once the modeler understands the states and state transitions in each object type or hierarchy, (s)he is able to analyze the model behavior at a conceptual level.
3. *Merge different decomposition levels and represent them as a whole:* Having handled each object type or hierarchy independently it is important that the different ORM schemas are merged into one complete model. In this merged model, all the decomposition levels can be viewed. After this step, SFD input parameters or values are easy to define because all existing states and transitions are already identified.

Finally, (4.) *Construct a stock and flow diagram and simulate the model:* This model should be inline with the resulting ORM decomposed scheme. Another important aspect in SFD models is simulation. Simulation allows continuous testing of assumptions and sensitivity analysis of parameters [50]. A simulation model distinguishes between stocks and flows. As stated earlier, stocks change overtime through flows to produce the dynamic behavioral patterns of the SFD model. To arrive at an SFD simulation model, the following guidelines may be of great importance:

a) *Represent each state category as a stock and use meaningful names:* After applying the decomposition mechanism, various levels of model abstraction can be seen. In each level there are elementary and compound states. Each of these states (compound or elementary) is depicted as a stock in SFD because they contain elements (objects) with similar properties. Compound states are made up of more than one elementary state. To improve understanding of the derived model(s), it is important to use simple and precise variable names [51].

b) *For each stock, define initial values or quantities:* For simulation computations, it is very important that defined initial values are appropriate. An initial value is a point at which quantities in a stock start to grow. For instance, if a stock has an initial value of 50, then in the simulation graph quantities in that stock start at 50 and if the initial value is zero, then the simulation of the stock also starts at zero.

c) *Identify flows (inflow or out flow) connecting to each stock:* Flows occur over a period of time. In business settings, there are several interactions and there exists many possible flow equations that are consistent with the Stock-Flow Diagram. But each problem domain has different variables and causal influences, the equations of the flows therefore must be entered or defined by the modeler. To successfully construct a Stock-Flow Diagram, it is necessary to understand the difference between stocks and flows [52]. Flows have rates at which quantities flow into or out of the stocks.

d) *For each added information link or converter, define input values or parameters:* Information links and converters are a central component in SFD models and play an important role. Information links can be difficult to model because of their abstract nature. Through information links, information from one converter/flow/stock can simultaneously flow to other SFD elements rapidly and with substantial twist. The addition of an information link can lead to profound impact on the model performance and simulation results [52]. It is therefore important that the modeler has a logical explanation for each information link added.

e) *Ensure that all input equations are logical and units are consistent:* SFD input equations or parameters should have a logical explanation. Each variable at the start of the connector should be included into the equations of the variable at the arrow tip of the connector. Secondly, the units on the right hand side of each equation should have the same unit measure as the ones on the left.

   Ensuring that all input equations are logical and units are consistent is a way of validating the internal structure of the SD model. Validity of the model in system dynamics means that the internal structure of the model is valid not its output behavior [53]. A valid SD model structure (the totality of the relationships between or among variables) can be used to test the effect of changes on the defined problem. A model that generates a behavior with little or no relationship with that of the system is most times unreliable and thus invalid. But if the model behavior replicates the behavior of the real system then it is valid.

f) *When evaluating the model, focus should be on interactions rather than individual elements:* This is because SFD elements influence each other through causal links and flows. If SFD elements are dealt with in isolation there would be a possibility of not arriving at a valid model because all elements in the model contribute to SFD model validity.

   Interactions or relations should be logical and before using the model to examine policy decisions, it should be validated against observed or likely trends. If the defined model reproduces or represents the real system (in the reference graph), then it is assumed to contain critical elements generating the problem; but if it does not reproduce the reference graph then the model structure and causal influences should be revised.

g) *Conduct tests:* In System Dynamics, modelers and users gain confidence in the model through testing. In [54], [55] three classes of tests are suggested: structure tests, behavior tests and policy implication tests. *Structure tests* determine how well the structure of the model matches the structure of reality, *Behavior pattern tests* determine how consistently model outputs match real world behavior; and *Policy implication tests* determine whether the observed system responses to policy changes replicate model predictions. In the process of conducting these tests, simulations depicting the behaviors of input variables are derived. To conduct these simulations, it is necessary to:
   - Choose appropriate simulation details when conducting test runs.
   - Use behavior over time graphs to understand the correlations between model variables.
   - Change input values to analyze the effect to change.

5 Running Example: Intrapartum Process

Intrapartum is the time from the onset of true labor until the delivery of the infant or placenta. For this example, we visited three Ugandan labor suites in Mukono Health Center, Kawolo Hospital and Mulago Hospital. We did so, to observe, note down the process, and record details on activities in the labor wards, e.g. doctor monitoring time, patient arrival time, number of patients received per day, activities in the labor ward, archival data, observe patients day to day behaviors. This was done for a period of three month.

The process: *A patient comes to the labor ward with her antenatal card from the antenatal clinic. She queues up. Her waiting time depends on a number of factors which are; her arrival time, the number of patients around and number of nurses on duty. When her turn comes, the nurse on duty takes her history and then examines her. This examination takes approximately 30 minutes for Mukono and Kawolo and 15-20 minutes for Mulago. The nurse establishes the patient's dilation stage. If the patient is 4cm dilated, she is admitted to the general ward. She only returns for examination if there is any complication or after 4 hours. During this time, after every 30 minutes monitoring of the labor progress, status of the mother and cervical dilation is done. When the patient is 8cm dilate, she is taken to the delivery room. While there, the nurse monitors descending of the head 2 hourly and the sticker. When the patient has 10cm dilate, she is ready to give birth. After delivery, she is taken back to the general labor ward. On discharge, the baby is taken for immunization.*

5.1 Constructing the Model

As a first ordering, below is a more structured textual description of this problem domain (a case "Intrapartum process in Ugandan Hospitals"):

A1 A patient comes to the labor ward with her antenatal card from the antenatal clinic.
A2 She queues up.
A3 Her waiting time depends on a number of factors which are; her arrival time, the number of patients around and number of nurses on duty.
A4 When her turn comes, the nurse on duty takes her history and then examines her.
A5 This examination takes approximately 30 minutes for Mukono and Kawolo and 15-20 minutes for Mulago.
A6 The nurse establishes the patient's dilation stage.
A7 If the patient is 4cm dilated, she is admitted to the general ward.
A8 She only returns for examination if there is any complication or after 4 hours.
A9 During this time, after every 30 minutes monitoring of the labor progress, status of the mother and cervical dilation is done.
A10 When the patient is 8cm dilate, she is taken to the delivery room.
A11 While there, the nurse monitors descending of the head 2 hourly and the sticker.
A12 When the patient has 10cm dilate, she is ready to give birth.

A13 After delivery, she is taken back to the general labor ward.
A14 On discharge, the baby is taken for immunization.

The next step is to extract the elementary sentences from this description. This set of sentences provides a complete basis to reason about (the essentials of) this domain.

1. We start with the main concepts that we immediately encounter in the domain description.
   - Person (Name).
   - LaborWard (Number).
   - Bed (BedNr).
   - Room (RoomNr).
   - DateTime (ddmmyy-hhmm).
   - Dilation (cm).

   These are rules that introduce entity types and the way they are identified. For instance, a Person is an entity type that is identified by the label type Name. Note that concepts are textually recognized by the capital first letter of their name. There are some sentences to express to indicate the kinds of persons considered:
   - Person is a medical person.
   - Person has a patient number.
   - Person is a newborn.

   All these sentences express unary facts about persons. We objectify these kinds of persons as sub concepts (subtypes) of the concept (entity type) Person:
   - MedicalPerson IS Person is a medical person.
   - Patient IS Person has a patient number.
   - Baby IS Person is a newborn.

   Patients also have an alternative identification:
   - Patient has PatientNumber.

   There are two kinds of medical persons; we describe and objectify them as follows:
   - MedicalPerson is a nurse.
   - MedicalPerson is a obstetrician.
   - Nurse IS MedicalPerson is a nurse.
   - Obstetrician IS MedicalPerson is a obstetrician.
   - MedicalPerson is on duty.
   - NurseOnDuty IS Nurse is on duty.
2. In terms of these concepts, we can describe the steps in the intrapartum process by the following elementary sentences:

   B1 Patient arrives at LaborWard at DateTime.
   B2 NurseOnDuty interviews Patient.
   B3 PatientQueuedUp IS Patient arrived BUT NOT interviewed.
   B4 Patient has AntenatalCard (Id).
   B5 AntenatalCard (Id) has PatientHistory (NotRefinedHere).
   B6 Nurse establishes Dilation stage from Patient.
   B7 AdmittedPatient IS Patient with Dilation $>$ 4.
   B8 DeliveringPatient IS AdmittedPatient with Dilation $>$ 8.
   B9 DeliveringPatient occupies Bed.
   B10 DeliveringPatient births Baby at DateTime.
   B11 MedicalPerson delivers Baby.
   B12 Mother IS DeliveringPatient with birth.
   B13 Nurse immunizes Baby.
   B14 MedicalPerson discharges Mother.
3. - Bed is in Room.
   - Room is in LaborWard.

These elementary sentences are shown graphically in the ORM scheme in Figure 6. Summarizing, the way of working to find the conceptual scheme is to start from an informal textual description, from which we extract the elementary sentence structures. The result is what is referred to as the information grammar, because it describes the syntax for elementary information exchange. The conceptual scheme is just a graphical representation of this language grammar.

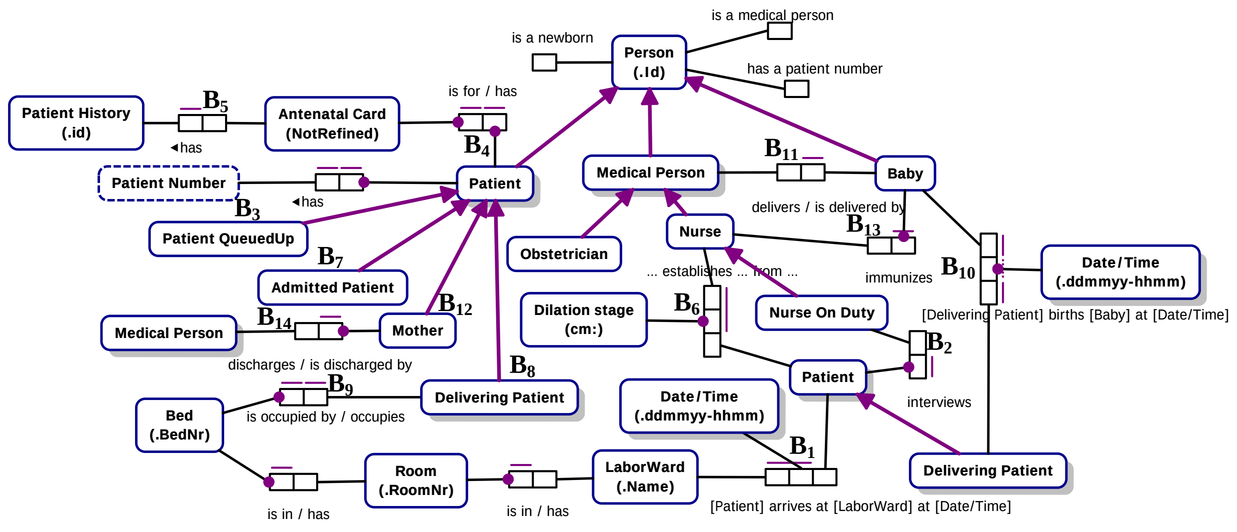


Figure 6. ORM Scheme for the intrapartum process in Ugandan Hospitals

5.2 Decomposing the ORM Model

Conceptual schemes for realistic applications tend to be rather large, which makes them hard to handle for human modelers. By applying decomposition, a complex conceptual scheme can be split up in a set of smaller conceptual schemes that each, in turn, is easier to handle. Decomposing an ORM model takes a number of steps (see Section 4.2).

5.2.1 Separate Object Types with Their Roles Into Unique ORM Schemes

An ORM model is made up of different constructs and different relating object types made up of different objects playing roles. To apply the decomposition mechanism, object types are separated into different ORM schemes. In cases where the ORM model has supertypes (object types with unique properties), each supertype is represented with a hierarchy of substates like in Figure 6. Each supertype is represented separately. This helps the modeler to represent states for each ORM schema with a unique identifier although the objects in each state have similar properties. Using Figure 6 as a basis, we have supertypes *Bed* and *Person*. In the life of a person, we have three elementary states: being *Baby, Patient* and *Medical person*. In the life of a bed, we have two elementary states; *Delivery Bed* and *Admission Bed*. The ORM sub-scheme for object types Person and Bed are represented in Figure 7. Object type Bed, has four different transitions $\{B_1,B_2,B_3,B_4\}$[5]. For state admission bed ($\mathcal{A}$), we have transitions (B_1,B_2) and for state delivery bed ($\mathcal{D}$), we have transitions (B_3,B_4). So $\mathcal{A}=\{B_1,B_2\}$ and $\mathcal{D}=\{B_3,B_4\}$.

Explanation: Initially, the bed is empty (B_1), when a patient is admitted, she occupies the empty bed (B_2). During delivery time the patient occupies the delivery bed (B_4) (in this case one patient occupies two beds the admission bed and delivery bed). The delivery bed is initially empty but when a patient is ready for delivery she occupies this delivery bed for a period of time before and after she is taken back to her admission bed (B_2), see the arrows between (B_1), (B_2), and (B_3), (B_4). In the running case study, the hospital(s) have limited resources, e.g. beds, medical personnel etc. Therefore a patient does not have the luxury of not occupying a bed when allocated one due to constrained resources. Note that in Figure 8, we do not distinguish between allocation of a patient to a bed and actual occupancy of a bed by a patient.

5.2.2 Handle Each Object Type Independently

In order to improve conceptualization of the SFD model, handle each object type independently.

[5] In [40] we discuss the state and transitions details

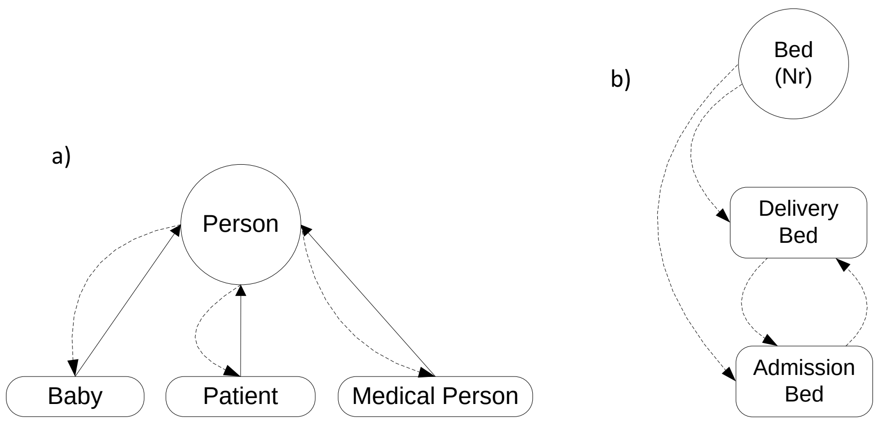


Figure 7. ORM sub-scheme for object types Person (a) and Bed(b)

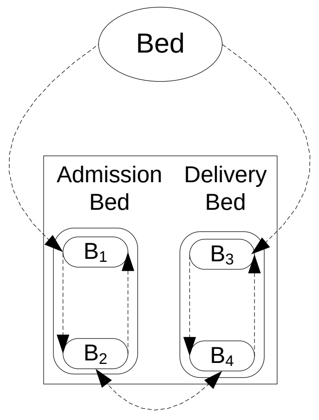


Figure 8. ORM sub-scheme for object type Bed

Doing so enables a modeler to understand how objects in each state relate to one another. For each object type;

1. Give each role a unique identification.
2. Categorize states into elementary and compound states.
3. Represent states in order of activity occurrence.
4. Add directed paths.

Once the modeler understands the states and state transitions in each object type or hierarchy, (s)he is able to analyze the model behavior at a conceptual level.

Example 1. When a patient arrives at the hospital, the nurse on duty records her details (*Patient History*). Here the state Patient History is initiated, it can only be updated when that patient next visits the hospital (see Figure 9). Patient History has two transitions '*is initiated*' and '*record*'.

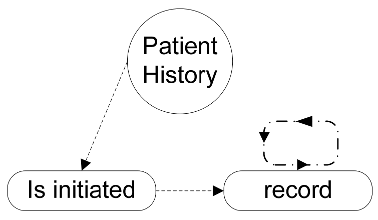


Figure 9. ORM sub-scheme for object type Patient History

In Figure 10, object type Patient, has four different states Patient Arrival ($\mathcal{A}$), Examination($\mathcal{X}$), Treatment($\mathcal{T}$) and Discharge($\mathcal{D}$). These states have transitions $\{P_1, P_2, \dots, P_9\}$.

Patient: At a higher level of abstraction the patient intrapartum starts with *Patient Arrival*. Then the patient is examined, leading to either *Treatment* or (if no treatment is required) *Discharge*. After treatment, the patient is discharged. In Figure 10 we can see this main structure, but the figure also displays the further details of the compound states.

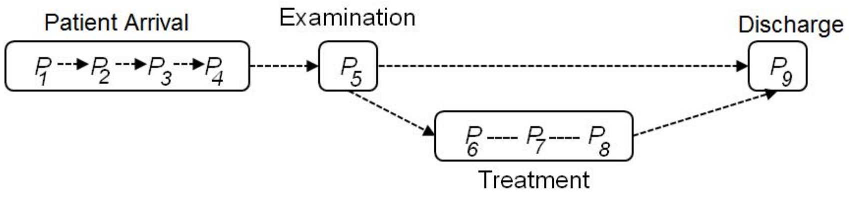


Figure 10. Detailed patient Intrapartum model

Explanation: When the patient arrives at the hospital, she queues up (P_1) then she is registered (P_2) by the nurse, after registration patient history is updated (P_3). We have: $\mathcal{A} = \{P_1, P_2, P_3, P_4\}$, $\mathcal{X} = \{P_5\}$, $\mathcal{T} = \{P_6, P_7, P_8\}$ and $\mathcal{D} = \{P_9\}$.

After recording patient history, the patient waits to be examined (P_4). When her turn comes she is examined (P_5), depending on the findings, she is either admitted (P_6) or discharged (P_9). If she is admitted (P_6), she is monitored (P_7) every 30 minutes until she gives birth (P_8). After giving birth she is discharged (P_9).

Medical Person(s) sub-states: The state *Medical Person*, has two base states *Nurses* and *Obstetricians*. These states are contained in four different complementary states *Medical Records* ($\mathcal{M}$), *Examination* ($\mathcal{X}$), *Treatment* ($\mathcal{T}$) and *Discharge* ($\mathcal{D}$) (see Figure 11). Some of these states are similar with the obstetrician's state. This is because in a hospital, if a case is very sensitive, for instance, an operation, it is handled by an expert who in this case is the obstetrician.

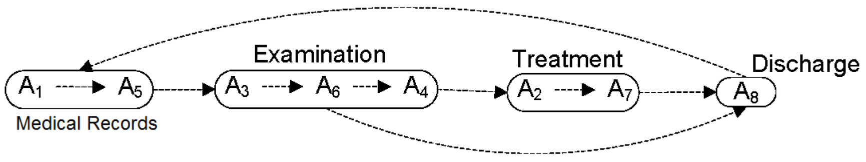


Figure 11. Nurse state model

Explanation: The life model of a *Nurse* has the following states; Administrates (A_1), Monitors (A_2), Examines (A_3), Admits (A_4), Records History (A_5), Establishes Dilation Stage (A_6), Births Baby (A_7) and Discharges (A_8). The life model of an *Obstetrician* has four states; Examines ($\bar{A}_3$), Admits ($\bar{A}_4$), Birth Baby ($\bar{A}_7$) and Discharges ($\bar{A}_8$). These states are contained in three different complementary states *Examination*, *Treatment* and *Discharge*. In Figure 11 and Figure 12 we present the life model of a nurse and an obstetrician respectively. We have $\mathcal{P} = \{A_1, A_5\}$, $\mathcal{X} = \{A_3, A_6, A_4, \bar{A}_3, \bar{A}_4\}$, $\mathcal{T} = \{A_2, A_7, \bar{A}_7\}$ and $\mathcal{D} = \{A_8, \bar{A}_8\}$.

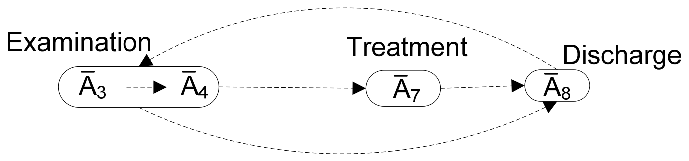


Figure 12. Obstetrician state model

5.2.3 Merge Different Decomposition Levels and Represent Them as a Whole

Having handled each object type or hierarchy independently it is important that the different ORM schemas are merged into one complete model. In Figure 13, we represent all the decomposition levels in the ORM schemes.

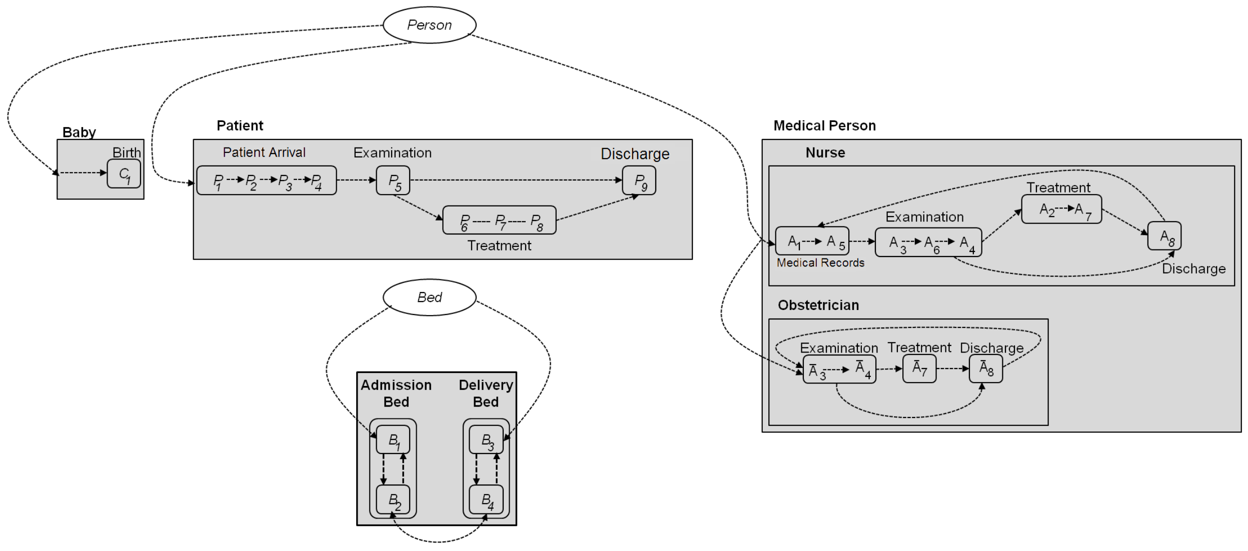


Figure 13. Decomposition levels in the ORM scheme shown in Figure 6

In this merged model, all the decomposition levels can be viewed. Understanding the state changes help the modeler define SD influencing transitions and input parameters. These influencing transitions and input parameters are discussed in Subsections 4.1. After this step, SFD input parameters or values are easy to define because all existing states and transitions are already identified.

6 Simulation

In this section, we discuss how an extended ORM scheme is converted into a stock and flow diagram. The ORM extensions we introduced, add essential System Dynamics concepts to the conceptual level description of ORM. As a result, the conversion is rather straightforward. This conversion focuses only on the states of object types and their transitions. The decomposition structure is ignored, since System Dynamics has no decomposition mechanism.

To construct the stock and flow model in Figure 14, we used the STELLA software package [56]. This package provides a practical way to dynamically visualize and communicate how complex systems really work; and to derive simulations. It, for instance, allows simulation ‘run’ systems over time, sensitivity analysis which reveals key leverage points and optimal conditions and partial model simulations which allows focus analysis on specific sectors or modules of the model. Results from the SFD are then presented as simulation in graph or table form. STELLA contains some aggregation operators like SUM, MIN, MAX etc. to combine values of two or more quantities. To enable simulation, mathematical equations for all model variables were defined thereby specifying the behavior of each model variable with exactly one equation. These equations depicted how defined variables change over time.

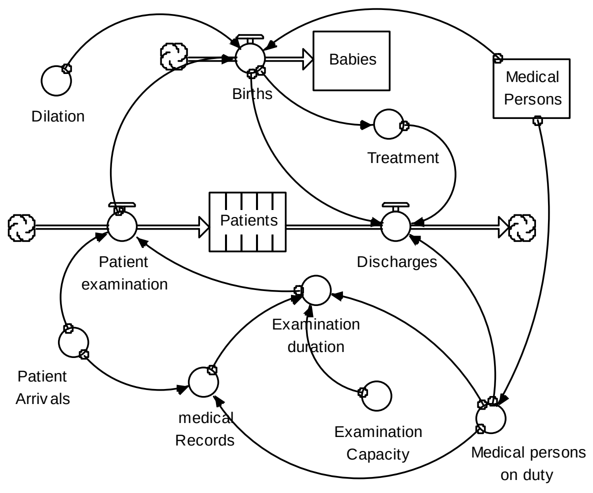
From the merged model in Figure 13, we have the following main decomposition levels: “Baby”, “Patient”, “Medical Person”, “Delivery Beds” and “Admission Beds”. [6] The decomposition levels in Figure 14 are depicted as stocks and in each there exists states and transitions i.e:

- “Patient Arrival”: $\mathcal{A} = \{P_1, P_2, P_3, P_4\}$

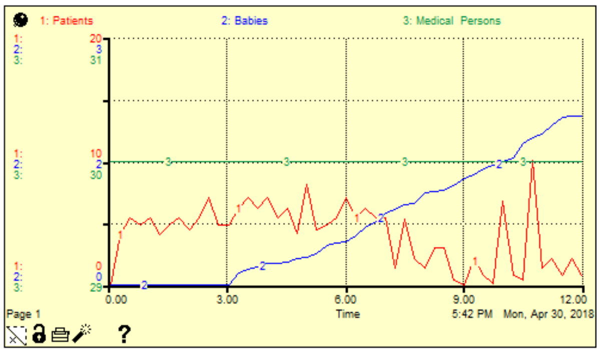
[6] To minimize on the complexity of the model, the Bed decomposition level and a number of transitions are excluded in this discussion.

- “Patient Examination”: $\mathcal{X} = \{P_5, A_3, A_6, A_4, \bar{A}_3, \bar{A}_4\}$
- “Treatment”: $\mathcal{T} = \{P_6, P_7, P_8, A_2, A_7, \bar{A}_7\}$
- “Discharge”: $\mathcal{D} = \{P_9,\}$
- “Medical Records”: $\mathcal{M} = \{A_1, A_5\}$

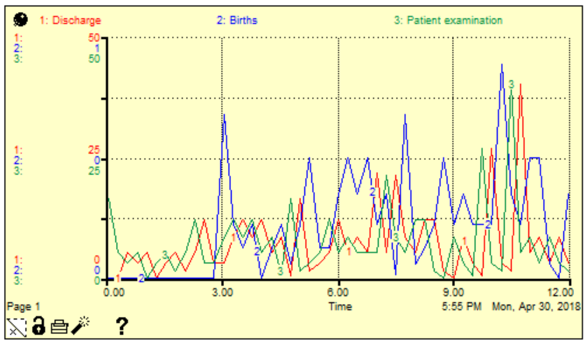
These states are categorized into flows and converters depending on the definitions stated in Section 3.2. For each decomposition level presented in Figure 13, we identified flows, i.e. inflow “Births” for stock “Babies”, and inflow “Patient examination” and outflow “Discharges” for stock “Patients”. The remaining states and some transitions are represented as converters (auxiliary variables). These were added because they influence the behavior of the quantities in stocks through flows. The equations defining the stock quantities are placed in the converters and flows.



***SFD 1:** Patient Life Stock and Flow Diagram representation*



Graph 1: Simulation Results for *Stocks Birth, Medical Person and Patients*



Graph 2: Simulation results for flows *Discharge, Births and Examined patients*

Figure 14. An SFD and resulting Simulations for a patient life model

In SFD 1, depicted in Figure 14, we see the stocks (“Patients”, “Babies” and “Medical Persons”), the auxiliary variables (“Dilation”, “Treatment”, “Patient arrivals”, “Medical records”, “Examination duration”, “Examination capacity” and “Medical persons on duty”) and the flows (“Patient examination”, “Births” and “Discharges”)

Inflows and outflows are rate variables with dynamic values, .i.e. they change over time. The behavior of flows is specified by the rate equation. In Figure 14, there are two inflows (“Patient examination” and “Births”) and one outflow (“Discharge”). The equation for inflow “Patient examination” is built upon two auxiliary variables “Patient arrival” and “Examination duration”. The equation for inflow “Births” is built upon one auxiliary variable “Dilation”, one inflow “Patient examination” and one stock “Medical persons”. The equation for outflow “Discharge” is built upon one inflow “Births” and two auxiliary variables “Medical persons on duty” and “Treatment”. The rate variables assigned to outflow “Discharge” control the decrease of stock “Patients”.

The defined auxiliary variables in Figure 14, consist of functions of stocks i.e. they constitute both static and dynamic input values. For instance, auxiliary variable “Examination duration” is influenced by three auxiliary variables “Medical persons on duty”, “Examination capacity” and “Medical records”. In the same model, there are auxiliary variables that influence other variables yet, they are not influenced by any variable in the model. For instance, auxiliary "Patient arrivals" influences inflow "Patient examination" and auxiliary "Medical records" and "Dilation" influences inflow "Births", but they are not influenced by any variable in the model.

In simulation results presented in Figure 14 SFD 1, stock "Medical Person" has a constant behavior because there is no direct inflow/outflow influencing it and also has a defined constant value (30). On the other hand, stock "Babies" is given an initial value of zero (0) and, from the simulation results, it initially has a delay. This is due to the delay defined in inflow "Births" but thereafter it depicts an exponential growth attributed to lack of an outflow. Stock "Patients" is also given an initial value of zero (0) with varying transit time, and an infinite inflow and capacity. In Figure 14 Graph 2, flows "Discharge", "Births" and "Patient examination" depict random variations. This is because both "Dilation" and "Patients arrival" occur at different time intervals. Variables "Dilation" and "Patient arrivals" directly influence inflows "Patient examination" and "Births" which also influence other variables in the model.

7 Conclusion

Describing object behavior at a conceptual level provides an effective means for domain understanding both by the domain expert and the system builder. In this article, we have combined ORM and SD in order to allow domain description in semi-natural language (in the ORM style). That way, the domain expert can validate the correctness of the domain description (its structure and its behavior) with a high degree of reliability. ORM and SD both have an expressive power such that even very complex domains can be handled. Thanks to the formal foundation of both ORM and SD, the semantics of such a description is sufficient to transform it into a computational model required for implementation. Here, we show how such a description is systematically transformed into a stock and flow description for a simulation in terms of the STELLA software package.

In Section 5, we have demonstrated this way of working by a non-trivial (but not too complex) case. We showed how improvement questions from policy makers can be represented into the final description, and this can automatically be presented in the simulation tool.

One of the main issues of this article was to show how the decomposition mechanism can be added to the conceptual language (thus extending ORM), and to argue how decomposition is an effective mechanism to master domain complexity. The intention of the decomposition mechanism is to help model more complex domains effectively. The state-of-the-art of advanced software packages such as STELLA do not (yet) support decomposition, therefore, during the transformation of the description into a computational model, the decomposition structure will not (directly) play a role.

Future research is required to further improve modeling of (very) complex domains in such a way that the domain experts can do this activity using their domain knowledge only. Another interesting activity would be to actually build a system that can automatically handle the systematic transformation of a domain description into a running simulation tool.

Evaluation and improvement of our theoretical founding will be a crucial aspect of our further research. Thus, obtained results will be made more valid, refined and more details added. Here we note that, this will not change the intention of the evaluated results; it will instead correct them, and make them more accurate.

Acknowledgements

We wish to thank the anonymous reviewers for the careful reading of our article, and the many valuable comments, that led to many improvements of the article.

References

[1] J. Duggan, “The case for personal data-driven decision making,” *Proceedings VLDB Endowment*, vol. 7, no. 11, pp. 943–946, 2014. [Online]. Available: https://doi.org/10.14778/2732967.2732969

[2] R. A. Buchmann and D. Karagiannis, “Modelling mobile app requirements for semantic traceability,” *Requirements Engineering*, vol. 22, no. 1, pp. 41–75, 2017. [Online]. Available: https://doi.org/10.1007/s00766-015-0235-1

[3] K. Siau and M. Rossi, “Evaluation techniques for systems analysis and design modelling methods-a review and comparative analysis.” *Information Systems Journal*, vol. 21, no. 3, pp. 249–268, 2011. [Online]. Available: https://doi.org/10.1111/j.1365-2575.2007.00255.x

[4] A. Hevner, S. March, J. Park, and S. Ram, “Design Science in Information Systems Research,” *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, March 2004. [Online]. Available: https://doi.org/10.2307/25148625

[5] R. Wieringa, “Combining Static and Dynamic Modelling Methods: A Comparison of Four Methods,” *The Computer Journal*, vol. 38, no. 1, pp. 17–30, November 1995. [Online]. Available: https://doi.org/10.1093/comjnl/38.1.17

[6] A. R. Da Silva, “Model-driven engineering: A survey supported by the unified conceptual model,” *Computer Languages, Systems & Structures*, vol. 43, pp. 139–155, 2015. [Online]. Available: https://doi.org/10.1016/j.cl.2015.06.001

[7] L. Grunske, L. Geiger, A. Zündorf, N. Van Eetvelde, P. Van Gorp, and D. Varro, “Using graph transformation for practical model-driven software engineering,” in *Model-driven Software Development*. Springer, 2005, pp. 91–117. [Online]. Available: https://doi.org/10.1007/3-540-28554-7_5

[8] H. Giese and R. Wagner, “From model transformation to incremental bidirectional model synchronization,” *Software & Systems Modeling*, vol. 8, no. 1, pp. 21–43, 2009. [Online]. Available: https://doi.org/10.1007/s10270-008-0089-9

[9] S. Brinkkemper, M. Saeki, and F. Harmsen, “Assembly techniques for method engineering,” in *Proceedings of the 10th International Conference on Advanced Information Systems Engineering*, ser. CAiSE ’98. London, UK, UK: Springer-Verlag, 1998, pp. 381–400. [Online]. Available: https://doi.org/10.1007/bfb0054236

[10] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, “On dynamic mode decomposition: theory and applications,” *arXiv preprint arXiv:1312.0041*, 2013. [Online]. Available: https://doi.org/10.3934/jcd.2014.1.391

[11] P. Wiśniewski, “Decomposition of business process models into reusable sub-diagrams,” in *ITM Web of Conferences*, vol. 15. EDP Sciences, 2017, p. 01002. [Online]. Available: https://doi.org/10.1051/itmconf/20171501002

[12] J. Sterman, *Business Dynamics- Systems Thinking and Modeling for a Complex World*, ed. McGraw Hill Higher Education, 2000. [Online]. Available: https://doi.org/10.1109/emr.2002.1022404

[13] J. Spector, D. Christensen, A. Sioutine, and D. McCormack, “Models and simulations for learning in complex domains: Using causal loop diagrams for assessment and evaluation.” *Computers in Human Behavior*, vol. 17, pp. 517–545, September–November, 2001. [Online]. Available: https://doi.org/10.1016/s0747-5632(01)00025-5

[14] J. Wang, “A review of Operations Research Applications in Workforce Planning and Potential Modelling of Military Training.” Edinburgh, Australia: DSTO Systems Sciences Laboratory, 2005.

[15] D. C. Lane, “Diagramming Conventions in System Dynamics,” *J. Oper. Res. Soc.*, vol. 51, no. 2, pp. 241–245, 2000. [Online]. Available: https://doi.org/10.1057/palgrave.jors.2600864

[16] J. Kleijnen, *Experimental design for sensitivity analysis, optimization, and validation of simulation models*. New York: John Wiley & Sons, Inc., 1998, ch. 6. [Online]. Available: https://doi.org/10.1002/9780470172445.ch6

[17] N. Melao and M. Pidd, “A Conceptual Framework for Understanding Business Processes and Business Process Modelling.” *Information Systems Journal*, vol. 10, pp. 105–129, 2000. [Online]. Available: https://doi.org/10.1046/j.1365-2575.2000.00075.x

[18] L. Luna-Reyes and D. Andersen, “Collecting and analyzing qualitative data for system dynamics: methods and models,” *System Dynamics Review*, vol. 19, no. 4, pp. 271–296, 2003. [Online]. Available: https://doi.org/10.1002/sdr.280

[19] F. Tulinayo, S. Hoppenbrouwers, and H. Proper, "Integrating System Dynamics with Object-Role Modeling," in *The Practice of Enterprise Modeling*, J. Stirna and A. Persson, Eds. Stockholm, Sweden: Springer Berlin Heidelberg, November 2008, vol. 15, pp. 77–85. [Online]. Available: https://doi.org/10.1007/978-3-540-89218-2_6

[20] A. t. Hofstede, H. Proper, and T. v. d. Weide, "Formal definition of a conceptual language for the description and manipulation of information models," *Information Systems*, vol. 18, no. 7, pp. 489–523, October 1993. [Online]. Available: https://doi.org/10.1016/0306-4379(93)90004-k

[21] T. Halpin and M. Curland, "Automated verbalization for ORM 2," in *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*. Heidelberg: Springer LNCS, 2006, vol. 4278, pp. 1181–1190. [Online]. Available: https://doi.org/10.1007/11915072_21

[22] L. Campbell, T. Halpin, and H. Proper, "Conceptual schemas with abstractions making flat conceptual schemas more comprehensible," *Data & Knowledge Engineering*, vol. 20, no. 1, pp. 39 – 85, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0169023X96000055

[23] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Programming*, vol. 8, pp. 231–274, 1987. [Online]. Available: https://doi.org/10.1016/0167-6423(87)90035-9

[24] J. Evermann and Y. Wand, "Towards ontologically based semantics for UML constructs," in *Conceptual Modeling - ER 2001*, ser. LNCS, S. Kunii, S. Jajodia, and A. Sølvberg, Eds. Springer Berlin / Heidelberg, 2001, vol. 2224, pp. 354–367. [Online]. Available: https://doi.org/10.1007/3-540-45581-7_27

[25] P. Frederiks and T. v. d. Weide, "Deriving and paraphrasing information grammars using object-oriented analysis models," *Acta Informatica*, vol. 38, pp. 437–488, June 2002. [Online]. Available: https://doi.org/10.1007/s002360200083

[26] P. van Bommel, P. Frederiks, and T. v. d. Weide, "Object-Oriented Modeling based on Logbooks," *The Computer Journal*, vol. 39, no. 9, pp. 793–799, February 1997. [Online]. Available: https://doi.org/10.1093/comjnl/39.9.793

[27] E. D. Kameni, T. P. van der Weide, and W. T. de Groot, "Natural model based design in context: an effective method for environmental problems," *Complex Systems Informatics and Modeling Quarterly (CSIMQ)*, vol. 12, pp. 86–116, 2017. [Online]. Available: https://doi.org/10.7250/csimq.2017-12.05

[28] M. M. Khamis and T. P. van der Weide, "Conceptual diagram development for sustainable egovernment implementation," *The Electronic Journal of e-Government*, vol. 15, pp. 33–43, 2017.

[29] M. M. Khamis and T. P. van der Weide, "A linguistic-based systematic approach to complex system dynamics and its application to e-government introduction in zanzibar," *Complex Systems Informatics and Modeling Quarterly (CSIMQ)*, vol. 11, pp. 85–111, 2017. [Online]. Available: https://doi.org/10.7250/csimq.2017-11.05

[30] T. M. Salvatore and F. S. Gerald, "Design and natural science research on information technology," *Decision Support Systems*, vol. 15, no. 4, pp. 251 – 266, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167923694000412

[31] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007. [Online]. Available: https://doi.org/10.2753/mis0742-1222240302

[32] T. Halpin and T. Morgan, *Information Modeling and Relational Databases*, 2nd ed. Morgan Kaufmann Publishers, 2008.

[33] G. Richardson and A. Pugh, *Introduction to System Dynamics Modeling with Dynamo*. Cambridge, MA, USA: MIT Press, 1981.

[34] T. Tao, *Analysis I*, 2nd ed., ser. Texts and Readings in Mathematics. Hindustan Book Agency, 2009, no. 37.

[35] L. Burmester and G. Matthias, "Combining System Dynamics and Multidimensional Modelling - A Metamodel Based Approach." in *Proceedings of the 14th Americas Conference on Information Systems*, Toronto, ON, Canada, August 2008.

[36] J. Leaver and C. Unsworth, "System Dynamics Modeling of Spring Behavior in the Orakeikorako Geothermal Field." *Elsevier Ltd*, vol. 36, no. 2, pp. 101–114, April 2007. [Online]. Available: https://doi.org/10.1016/j.geothermics.2006.08.001

[37] K. Tan, M. Ahmed, and D. Sundaram, “Sustainable Enterprise Modelling and Simulation in a Warehouse Context,” in *Business Process Management Journal*. Emerald Group Publishing Limited, 2010, vol. 16, pp. 871–886. [Online]. Available: https://doi.org/10.1108/14637151011076511

[38] F. Tulinayo, P. Van Bommel, and H. Proper, “Enhancing the system dynamics modeling process with a domain modeling method,” *International Journal of Cooperative Information Systems*, vol. 22, no. 02, p. 1350011, 2013. [Online]. Available: https://doi.org/10.1142/s0218843013500111

[39] H. Proper and T. v. d. Weide, “Information disclosure in evolving information systems: Taking a shot at a moving target,” *Data and Knowledge Engineering*, vol. 15, pp. 135–168, 1995. [Online]. Available: https://doi.org/10.1016/0169-023x(94)00035-d

[40] T. van der Weide, F. P. Tulinayo, and P. van Bommel, “Static and dynamic aspects of application domains: An inductively defined modeling technique that allows decomposition,” *Complex Systems Informatics and Modeling Quarterly*, no. 7, pp. 25–50, 2016. [Online]. Available: https://doi.org/10.7250/csimq.2016-7.02

[41] H. Balsters, A. Carver, T. Halpin, and T. Morgan, “Modeling dynamic rules in ORM,” in *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, ser. LNCS, R. Meersman, Z. Tari, and P. Herrero, Eds. Springer Berlin / Heidelberg, 2006, vol. 4278, pp. 1201–1210. [Online]. Available: https://doi.org/10.1007/11915072_23

[42] H. Balsters and T. Halpin, “Formal semantics of dynamic rules in ORM,” in *Proceedings of the OTM Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: 2008 Workshops: ADI, AWeSoMe, COMBEK, EI2N, IWSSA, MONET, OnToContent + QSI, ORM, PerSys, RDDS, SEMELS, and SWWS*, ser. OTM ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 699–708. [Online]. Available: https://doi.org/10.1007/978-3-540-88875-8_94

[43] M. Albrecht, “Introduction to Discrete Event Simulation,” 2010.

[44] G. Lee, “Towards an integration of computer simulation with computer graphics,” in *Proceedings of the Western Computer Graphics Symposium*, 1999.

[45] Z. Kowalczuk, “Discrete approximation of continuous-time systems: a survey,” *IEE Proceedings-G*, vol. 140, no. 4, pp. 264–278, 1993. [Online]. Available: https://doi.org/10.1049/ip-g-2.1993.0045

[46] E. Pruyt, “Dealing with uncertainties? Combining System Dynamics with Multiple Criteria Decision Analysis or with Exploratory Modelling citation,” in *Proceedings of the 25th International Conference of the System Dynamics Society*. Boston, MA, USA: System Dynamics Society, 2007.

[47] M. Schaffernicht, “Causal loop diagrams between structure and behaviour: A critical analysis of the relationship between polarity, behaviour and events,” *Systems Research and Behavioral Science*, vol. 27, no. 6, pp. 653–666, April 2010. [Online]. Available: https://doi.org/10.1002/sres.1018

[48] R. D. Arnold and J. P. Wade, “A definition of systems thinking: a systems approach,” *Procedia Comput Sci*, vol. 44, pp. 669–678, 2015. [Online]. Available: https://doi.org/10.1016/j.procs.2015.03.050

[49] F. Tulinayo, P. van Bommel, and H. Proper, “From a system dynamics causal loop diagram to an object-role model: A stepwise approach,” *Journal of Digital Information Management*, vol. 10, no. 3, pp. 191–203, 2012.

[50] J. Morecroft, “System Dynamics and Microworlds for Policymakers.” *European Journal of Operations Research*, vol. 35, no. 3, pp. 301–320, June 1988. [Online]. Available: https://doi.org/10.1016/0377-2217(88)90221-4

[51] C. Caulfield, D. Veal, and S. Maj, “Implementing System Dynamics Models in Java,” *International Journal of Computer Science and Network Security*, vol. 11, no. 7, pp. 43–49, July 2011.

[52] C. Kirkwood, *System Dynamics Methods: A Quick Introduction.* Arizona State university, College of Business, 1998.

[53] Y. Barlas and S. Carpenter, “Philosophical roots of model validation: Two paradigms,” *System Dynamics Review*, vol. 6, no. 2, pp. 148–166, Summer, 1990. [Online]. Available: https://doi.org/10.1002/sdr.4260060203

[54] Y. Barlas, “Multiple tests for validation of system dynamics type of simulation models,” *European Journal of Operational Research*, vol. 42, no. 1, pp. 59–87, 1989. [Online]. Available: https://doi.org/10.1016/0377-2217(89)90059-3

[55] Y. Barlas, “Formal aspects of model validity and validation in system dynamics,” *System Dynamics Review*, vol. 12, no. 3, pp. 183–210, Autumn (Fall), 1996. [Online]. Available: https://doi.org/10.1002/(sici)1099-1727(199623)12:3⟨183::aid-sdr103⟩3.3.co;2-w

[56] K. Saeed, “An academic user’s guide to stella barry richmond, steve peterson, and peter vescuso lyme, nh: High performance systems, inc., 1987,” *System Dynamics Review*, vol. 5, no. 2, pp. 217–220, 1989. [Online]. Available: https://doi.org/10.1002/sdr.4260050213