A Methodology for Eliciting and Modeling Exceptions

Mor Peleg, PhD[a], Judith Somekh, M.Sc.[b], and Dov Dori, PhD[b]

[a]Department of Management Information Systems, University of Haifa, Israel, 31905

[b]Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa 32000, Israel

All communication should be with:

Judith Somekh

POBox 150, Kerem Maharal 30840, Israel

yuditol@technion.ac.il; ysomekh@gmail.com

Fax: 1 (413) 375-3755

Abstract

Exceptions in safety-critical systems must be addressed during conceptual design and risk analysis. We developed a conceptual model of exceptions, a methodology for eliciting and modeling exceptions, and templates for modeling them in an extension of the Object Process Methodology (OPM) – a system analysis and design methodology and language that uses a single graphical model for describing systems, including their timing exceptions, which has been shown to be an effective modeling methodology. Using an antibiotics treatment guideline as a case study, we demonstrate the value of our approach in eliciting and modeling exceptions that occur in clinical care systems.

Keywords— Systems specification methodology, Exceptions, Conceptual modeling, Elicitation methods.

I INTRODUCTION

Modern systems in general and medical systems in particular are becoming ever more complex, increasing the number of exceptional situations they have to cope with. The need to anticipate possible exceptions and specify exception-handling processes in medical systems is particularly critical, as these systems must ensure the patient's safety. For example, decision-support systems that recommend medical treatment should be aware of life-threatening side-effects and offer advice on handling such situations. Systems that let physicians prescribe medications need to be able to check for contraindications and drug-drug interactions. There is a shortage of methods for supporting system architects and designers in the analysis and modeling of exceptions as an integral part of the system's conceptual modeling process. In view of this state of affairs, we propose a methodology for eliciting and modeling exceptions.

The cost of correcting errors or implementing new requirements discovered during coding is between 5 to 10 times higher than the cost of correcting errors discovered during the requirements phase, and the cost of correcting errors discovered during the maintenance phase is between 100 and 200 higher [1]. There is tremendous potential to save these costs and associated time resources through improving requirements and exceptions modeling. Such improvements can be achieved by developing and adopting methods that help elicit a wide range of exceptions and incorporate them into the conceptual model during the early design phase of a system's lifecycle.

Our goal was to develop a methodology that would assist systems analysts and knowledge engineering in eliciting a wide range of possible exceptions and modeling them in a standard way. To cover such a desired wide range of exceptions, we chose to define exception in a broad way. There are many definitions of exceptions [2-4]. Klein and Dellarocas [5] defined an

exception as "any deviation from an “ideal” collaborative process that uses the available resources to achieve the task requirements in an optimal way", or, in other words, "any departure from a process that achieves the process goals completely and with maximum efficiency". Influenced by this definition, we define an exception as an occurrence that deviates from the "ideal", normal flow, as defined by the system designer, and causes a change in the primary goal of the current process, possibly leading to a change in the entire system's goal priorities. The ideal, normal-course system model makes many explicit and implicit assumptions that can be defined as conditions on processes imposed before, during, and after system activities take place. Violations of these conditions are exceptions.

The design of exceptions and their handling mechanisms requires the understanding of a wide range of potential exceptions, as well as the ability to represent them via explicit modeling constructs grounded in a solid methodology. Object-Process Methodology (OPM) [6] is a modeling methodology that specifies the behavioral and structural aspects of a system in a single model. OPM is especially suitable for modeling dynamic systems, as it can directly express events, event-condition-action (ECA) rules, guarding conditions (i.e., preconditions that guard the execution of a process), and timing exceptions [7]. In its current form, however, OPM lacks the ability to model the full range of exceptional behaviors, such as asynchronous or non-temporal exceptions, neither does it offer templates for simplifying and standardizing the specification of different exception types.

In this paper, we present exceptions methodology that comprises several elements. The first is a conceptual model of exceptions, represented in OPM. The second is a set of exceptions elicitation guidelines that are independent of a modeling formalism. The third element is an OPM-based extension for modeling asynchronous exceptions, and the fourth is a set of exception

templates for modeling exceptions in a standard way using OPM, along with guidelines for using the templates. We demonstrate the application of our exceptions methodology to modeling an antibiotics treatment clinical guideline.

II BACKGROUND

Our exceptions modeling methodology leverages related work on models for exceptions and their handling, along with works on exceptions classification. Integrating and extending these works, we define a conceptual model of exceptions and use characteristics of exception components to classify exceptions. As we formulate our conceptual model of exceptions using OPM, we start this section with an introduction to OPM and the reasons that we chose this methodology. We then review related work and introduce our OPM model of exceptions.

A. Object-Process Methodology

Taught in various institutions of higher education and applied in a variety of industry sectors, Object-Process Methodology (OPM) [6, 7] is a holistic, integrated approach to the design and development of systems. Underlying OPM is a philosophy stipulating that to faithfully and naturally analyze and design systems in any domain, processes, like objects, should be considered as stand-alone "things" and not encapsulated within objects as in the standard object-oriented (OO) approaches. Objects in OPM are physical or informational things that persist, while processes are transient things that transform objects. Processes can transform objects by affecting their state, generating new objects, or consuming existing objects. OPM models are represented graphically as Object-Process Diagrams (OPDs), which are automatically-translated into sentences in a subset of English, known as Object-Process Language (OPL). For example, **System** is one of the objects depicted in Figure 1; the **Exception Handling** process affects the **System**, as shown by the effect link connecting these entities. The fourth sentence in the OPL

specification, shown in Figure 1, conveys the same specification.

Objects and processes are linked via structural links and procedural links. A structural link represents a persistent relationship between two objects (e.g., Triggering Object **triggers** Trigger, shown in Figure 1). A procedural link connects an object to a process in which it participates. A procedural link can be (1) an enabling link—an instrument or agent link, which connects a process to an enabler—an object acting as an instrument or an agent that needs to be present for the process to occur (e.g., **Trigger** and **Guarding Condition** are instruments of the **Exception Handling** process, shown in Figure 1), or (2) a transformation link, which connects the process to a transformed object (i.e., an object which changed its state, was created, consumed, or affected). The compact OPM symbol set, its syntax, and its semantics are provided in [8]. We also explain OPM symbols in figures where they are used.

The complexity of systems is managed in OPM models by abstraction-refinement mechanisms, notably out- and in-zooming, which can be used to hierarchically expose or hide details of objects and processes. This way, a top-level view of the system is expanded into a set of increasingly detailed diagrams that provide the details of the processes and objects shown in the top-level view. OPM is supported by OPCAT [9], a modeling software environment that has been tested and successfully applied in academia and industry that is used in this work to model the case study with its exceptions.[1]

We selected OPM as the language for modeling exceptions within the system model due to its following favorable features.

(1) OPM's visual representation and its corresponding OPL textual representation may aid users in understanding the system models.

[1] The academic version of OPCAT can be downloaded from www.opcat.com

(2) OPM supports representation of system dynamics, and it includes the basic building blocks for representing exceptional behavior: triggering events, guarding conditions, timing constraints, timing exceptions, and flow-of-control. These features are the basic elements required for modeling exceptional behavior. For example, in Figure 1, **Trigger** is specified as the event triggering the **Exception Handling** process (symbolized by the letter 'e' next to the **Exception Handling** process) and **Guarding Condition** is specified as a guarding condition of that process (specified by the letter 'c' next to the **Exception Handling** process).

(3) OPM was shown by our group [7] to be significantly better in specification quality, compared with OMT, the main predecessor of Unified Modeling Language (UML), which is the Object Management Group's software systems analysis and design industrial de-facto standard. OPM's effectiveness stems from its single model, which explicitly represents in single model different process-related aspects, such as triggering events of a process and the objects participating in a process. When multiple-model languages, such as UML, are used, details need to be gathered by the user from multiple model views, such as dynamic, functional, and object models.

(4) Through its recursive seamless complexity management (abstraction-refinement) mechanisms, OPM is especially geared to managing systems' complexities. Using these mechanisms, exceptional processes and their recovery parts can be modeled at a fine detail level, but triggered and handled at a coarse detail level, so they do not clutter the main, normal-course process in the OPM system model.

(5) OPM's modeling tool, OPCAT, supports the notion of meta-libraries [10], which are OPM models that are imported into another OPM model (representing a system). This way, the objects and processes of the meta-library can be used as design patterns, templates that are utilized

within a system model. The system's objects and processes can be tagged and classified as performing roles of objects and processes of the meta-library.

B. Models for exceptions

Exceptions have been mostly researched in the non-medical domains of real-time and embedded systems. Researchers have tried to model exceptions, either through embedded approaches that integrate the exceptional semantics within the model of normal system behavior, or through separated approaches [4] that detach the system's exceptional semantics from its normal behavior. Most of these separated approaches adopted Event-Condition-Action (ECA) rules as a method for modeling exceptions. In this approach, a triggering event, called *triggers* trigger an *exception*, and when certain *guarding conditions* hold, an *exception-handling action* or process takes place. As an example, an anaphylactic shock can serve as a trigger, such that if the patient has recently started taking Penicillin, this trigger invokes the action of administering Adrenaline. Figure 1 is an Object-Process Diagram (OPD), with its equivalent automatically-generated OPL paragraph, depicting the conceptual model of exceptions that captures the ECA paradigm. As Figure 1 shows, an **Exception** object has two (object) parts – a set of **Triggers** and a **Guarding Condition**. An **Exception** is further characterized by an **Exception Handling** process. This process requires the **Trigger** and **Guarding Condition**. These are represented by the instrument links (white lollipops) that connect them to the **Exception Handling** process annotated with an 'e' and 'c' inside the lollipop circle to indicate the roles of event and guarding condition, respectively. The **Exception Handling** process affects the **System**. Whereas the ECA paradigm abstracts away from the source of the triggering event, our model includes a **Triggering Object** that triggers the **Trigger**, as conveyed by the general structural link between **Triggering Object**

and **Trigger**. It is important to consider the triggering entities during system analysis, because these entities are sources of events, which may be exceptional, and therefore should be observed [11].

Some researchers extended the ECA approach by introducing other considerations. Justified Event Condition Action rules [12] add justifications—a specification of the context where the rule will be performed—in order to capture more contexts in workflow modeling and to deal with uncertainties involved in organizational processes. In Adaptflow [11], a consultation system that can handle exceptional situations of changed patient condition, such as infection or toxicity, the ECA rules are extended with an optional valid time part that specifies a time period during which the modification action need to be applied. The ECA rules can be executed as active rules and integrated with workflow management systems [11, 13, 14].

While ECA paradigms are often specified as rules, some languages [4, 15-18] use graphical symbols to specify exceptional events and mechanisms for handling them. In all these models, the graphical specification of exceptions is integrated into a graphical specification of a process model. The objective of this integration is to facilitate comprehension and design of exceptions and their handling as they are composed with the normal process execution, avoiding a dedicated means for modeling exceptions. In the OPERA [17] project, the graphical specification is later compiled into a rule-based language.

Unlike the formalisms discussed above, which allow abnormal system execution by representing normal and abnormal behavior explicitly, Mulyar et al. [19] proposed a declarative formalism, called CIGDec, which allows flexible execution of processes by specifying only the minimal set of constraints between tasks that must not be violated. Any behavior that satisfies these constraints be it normal or abnormal behavior, such as behavior in emergency situations, can be

executed. Using a set of tasks and relations among them, their declarative model specifies what to do without the need to specify a particular order of the normal and the exceptional tasks in all the possible scenarios, as required by imperative approaches. Their approach can meet the flexibility requirement in medical environments that feature various expected and unexpected exceptions, overcoming problems encountered while using imperative languages for modeling clinical guidelines. However, this approach can mask the normal or ideal process flow, rendering it unsuitable for rigid processes with strict sub-processes order.

Much research has been devoted to the handling of exceptions [4, 5, 14, 20-24], including mechanisms for detecting, diagnosing, and resolving exceptions, including the management of the exception and resuming to normal system operation [4, 5]. Further discussion of these works is beyond the scope of this paper, as we focus on elicitation and modeling of exceptions rather than on characterization of exception handling processes.

C. Dimensions of Exceptions

Many researchers explored the different types of exceptions that may occur in systems [2, 4, 5, 16, 18, 21, 23-28], as shown in Table 1.We integrated the taxonomies defined in these works with our insights from analyzing industrial case studies from the healthcare and embedded real-time systems domains to create a new conceptual model of exceptions. As shown in Figure 1, the main components of our exceptions conceptual model are the objects that participate in Exception Handling processes, namely **Triggering Object**, **Trigger**, **Exception**, and **Guarding Conditions**. The characteristics of these objects form the dimensions of our classification of exception types. Each exception can be characterized as having a specific value for each dimension. The dimensions of exception are described in the rest of this subsection. Table 1

summarizes research works dealing with the various dimensions of exception and their research foci.

Some dimensions are interdependent, making some attributes seem semantically redundant. We note such cases in the discussion below. Since we developed the exception dimensions for the purpose of eliciting as many exceptions as possible, it is important to consider all the dimensions, as thinking about the same phenomenon from different viewpoints may help modelers elicit more exceptions. For the purpose of modeling exceptions using standard patterns, the redundancies are not useful and therefore a single template may be used to model the redundant information about an exception.

1. Triggering Object

The **Triggering Object** of the **Exception** exhibits the following attributes, each with its own set of values:

- ***Origin*** – an attribute denoting the source of the triggering entity, which can be **systemic** (internal), if the triggering is due to a failure of a system component or process (e.g., a patient's side effects), or **environmental** (external) [21]. An example of an external event is when an exceptional laboratory test result becomes available and may be observed by a decision-support system. This dimension is determined by the boundary of the system, as envisioned by the system designers.
- ***Essence*** – an attribute denoting to the essence of the triggering entity, which specializes into **Human** and **Non-human** [27].

The dimensions related to **Triggering Object** are shown graphically in Figure 2. The full OPD-set that graphically shows the characteristics of the exceptions' conceptual model is shown elsewhere [29]. In the rest of this subsection, we provide a textual description of these

characteristics.

2. Trigger

The trigger of the exception exhibits the following eight attributes, each with its set of values:

- ***Synchronicity*** [30] – A trigger is an ***asynchronous event*** if it can occur at any point in time, independent of system processes (e.g., a sudden side effect). A trigger is a ***synchronous branch*** if it occurs at a specific branch point in a process. At the branch, a decision is made to follow a normal or an exceptional path. A path includes a serial flow of processes that can be executed in order to achieve some goal. A path is considered exceptional if it is below a threshold defined as favorable for meeting the process goal. An example of a synchronous path is taking the drug with alcohol, after the drug is prescribed.
- ***Condition Violations –***Each "ideal" process can have a set of preconditions, post-conditions and invariants that mandate its correct execution. Any violation in these conditions can trigger an exception. Preconditions determine whether the task is allowed to start execution, post-conditions determine whether the task is permitted to successfully commit. Finally, invariants are syntactically like preconditions and post-conditions but they are enforced during the execution of the task or process [31]. Pre- and post-condition violations are associated with synchronous triggers whereas invariant violations are associated with asynchronous triggers.
- ***Predictability*** – an attribute denoting whether the trigger can be (i.e., has the value) **expected** if it can be foreseen at design time, or **unexpected** otherwise. Our work considers expected exceptions.
- ***Measurability*** – The ability of the trigger to be **measured**, which can be measurable, or **immeasurable**. A trigger with **Measurability** of value **measurable** is triggered when a

specified measurable threshold value is reached or exceeded or (for time) elapsed, or some numeric boundary value is violated. As shown in Figure 3, the threshold value is expressed in measurement units, which are composed from basic measurement units, including temperature, length, charge, time, mass, angle and luminous intensity [32]. A timing trigger [30] is an example of a **measurable Trigger**. The measurement process itself, which must be performed before each such Trigger, can be single, i.e., occur once, or periodically. An immeasurable Trigger cannot be quantified by means of measurable units. An example for this kind of trigger is failure of a network connection. Such triggers are often binary – either something happens or not.

- ***Cause*** – an attribute denoting the cause of the exception, and which specializes into **Human Cause** and **Non-human Cause**. As noted by Fridsma and Thomsen [25], who researched exceptions in the medical domain by simulating the process of medical care within organizations, Human **Cause** can be error, malicious action, or non-compliance. While usually, non-compliance in the medical domain is attributed to patients, errors (exceptions) due to non-compliance of care providers also occur. This topic has been studied by Quaglini et al. [26]. Klein and Dellarocas [5] have identified other human causes of exceptions, including suboptimal coordination or communication. Other human-related causes include an "act of nature" that causes the human to dysfunction in some way outside his control and without involving his intellect (e.g., adverse effects to drugs). There are many **Non-human Causes** [16], such as organizational causes [11] (e.g., resource unavailability), work item failure (e.g., problem with a laboratory test), deadline expiration [16], hardware or software incompatibility. Note that human causes could only occur for Triggering Entity of human essence and non-human causes could only occur for Triggering Entity of non-human essence.

- ***Detection Scope –*** Detection Scope concerns the scope or space containing the points at which the Trigger can occur. The Detection Scope of an exception (i.e., failure or exceptional occurrence) [4, 30] can be divided into three levels: task-level Exception (a failure related to one specific task), block-level Exception (failure that can occur within some block of tasks), and system-level Exception (failure that is related to the entire system and can happen in each task within each block). System-level exceptions are global occurrences that may possibly affect every process, and for which the reactions may be defined at the system's global level and possibly refined for specific processes if different policies need to be adopted. Unavailability of a resource is an example of a system-level trigger.
- ***Frequency*** – The trigger can be **frequent** or **rare**; **frequent** exceptions tend to be part of the normal flow and are usually embedded in the model [4]. Exceptions whose frequency is **rare** will be modeled usually in a separate procedure.

3. Guarding Conditions

Guarding conditions are preconditions that must be met for the Exception-handling process to take place. Guarding conditions have one attribute called **Complexity**.

- ***Complexity*** of guarding conditions can be **atomic** or **compound**. A Guarding Condition whose **Complexity** is **atomic** has only a set of states. A guarding condition whose Complexity is **compound** consists of two or more atomic guarding condition with one or more logical statements that combine states of atomic guarding conditions using logical operators.

4. Exceptions

Exceptions are characterized by the attributes **Effect Scope** and **Severity**.

- ***Effect Scope*** is similar to the ***Detection Scope*** of a trigger (described above), but the scope

here relates to the effect of the exception or of its resolution process on current or future systems processes.

- ***Severity*** [33] of an exception is determined by its potential effect on the normal operation of the systems, which can be ***ignorable,*** when it does not affect the normal operation of the system, ***light***, when it does not cause any error, but a continuous supervision of the faulty component is required and execution of a recovery function may be required as well, ***true***, when the exception causes malfunction of main system components, but replaceable components or other ways of recovery are possible, which enable reaching the original goal, or ***hard***, when the normal operation of the system is not possible, because main system components are malfunctioning or recovery from crisis has failed. The goal cannot be reached in the current setting, usually resulting in the termination of the current task.

5. Exception Handling

A common theme to the works regarding exception handling [4, 5, 14, 20-24] is the identification of three phases in exception handling: exception detection, diagnosis (or analysis), and resolution, which includes managing the exception and resuming to normal system operation [4,5]. Our conceptual model for exception handling is shown in Figure 4. Zooming into the Exception Handling process shows that it consists of three sub-processes: Detecting, analyzing, and Resolving. The OPD shows the objects that participate (serve as instruments) in the different sub-processes. It also shows how these sub-processes change the state of the Exception Instance (Instance of Exception, symbolized by the Instantiation link) from "active" to being resolved ("resolving") and finally to "resolved".

III EXCEPTION TEMPLATES

Based on our conceptual model of exceptions and their dimensions, we defined exception templates for the different types of exceptions that allow representing exceptions graphically as part of the system design specification. Our approach is embedded and is based on the ECA approach using a graphical notation, which is an extension of OPM.

Our exception template models can be integrated into OPM system models as predefined, reusable building blocks. Similar to design patterns [10], exception templates help keep the design standardized and minimize the mental reinvention workload during the system modeling and design phase.

We did not attempt to provide a different template for each possible combination of dimension values. Instead, we chose several primary dimensions for which we defined different templates. The other dimensions are used to refine and customize the main patterns, as explained below.

Our exception templates rely on the notion that exceptions can be synchronous or asynchronous and result from condition violations. Hence, while modeling exceptions, the designer should seek violated conditions, as these trigger the exceptional situation that requires exception handling.

Like design patterns, exception modeling templates are divided into structural templates and behavioral templates. Structural templates define various condition sets that can be incorporated into the behavioral templates, whereas the latter define various flows of exceptional situations along with their handling. Both are described in detail below.

A. Behavioral Templates

Viewing **Trigger** as a condition violation, behavioral exception templates are divided into three types, corresponding to the three types of **Trigger** condition violations (presented in Section II.C.2). Each behavioral template can be customized to suit the specific exception's characteristics. The customization of the behavioral template is obtained via a set of values that

are assigned to the various attributes (dimensions) of exception and by refining the behavioral templates with the structural templates and other relevant OPM constructs, as exemplified in Section V.

1. Synchronous Condition Set Violation Templates

There are two types of synchronous condition violation templates: precondition and post-condition set violations. The **Precondition Set Violation** template defines a mechanism for modeling synchronous exceptions that result from the violation of a system process preconditions (see Figure 5a). If the **Precondition Set** is **fulfilled**, the flow of control synchronously and serially branches to **Normal Processing**. Otherwise, i.e., when **Precondition Set** is **violated**, it branches to **Exception Handling**. The **Post-condition Set Violation** template is the same as the previous template, except that it represents violations of post-conditions (see Figure 5b).

In each one of the behavioral templates, when contextually appropriate, the links between the processes and the condition-set objects can be replaced by transformation links, condition links, enabling links, or event links. For example, one or two of the consumption links in the **Post-condition Set Violation** template (Figure 5b) can be changed into instrument links.

2. Asynchronous (Invariants Set) Violation Templates

Asynchronous violation templates (also referred to as Invariant Set Violation templates) define a mechanism for modeling exceptions that result from the violation of one or more invariants (constraints) associated with the process being executed. There are three such templates.

The first two templates, presented in figure 6a and 6b, describe **Halting Exceptions**—exceptions that halt the execution of the normal process and instead, invoke an exception-handling process. In this model, the original goal of the normal process is replaced with the goal

of exception handling. To stress the fact that the violation halts the execution of the normal process and invokes execution of the exception-handling process, the OPM timing exception link (from **Normal Processing** to **Exception Handling**) is extended to expresses a non-temporal exception that causes control to pass to another process (**Exception Handling**). After **Exception Handling** completes its execution, if applicable (i.e., if the initial original goal can potentially still be reached), resuming normal execution should be modeled as an addition to the template. The template shown in Figure 6a explicitly models the **Invariants Set**. The exceptional violation of invariants during the execution of **Normal Processing** causes the **Invariants Set** of **Normal Processing** to enter the **violated** state, which triggers the execution of **Exception Handling**. The template shown in Figure 6b models an event triggered by a triggering object that triggers the **Exception Handling** process. The invariant violation is expressed here implicitly, assuming that the specific event is not expected to occur during the "ideal" execution of **Normal Processing**. The third template, presented in Figure 7, is equivalent to the previous one, except that the triggered exception results in augmenting the current system's goals by adding the goal of handling the exception. In this case, **Exception Handling** is executed along with **Normal Processing**. This template usually concerns exceptions with **ignorable** or **light Severity**, where no halting of the normal process is required.

B. Structural templates

Structural templates are related to refinements of condition sets. We provide templates for two such refinements: measurable and compound condition sets. We explain the first template below. The template for specifying complex guarding conditions includes the ability to encapsulate complex logical statement and is explained and demonstrated in [29].

The **Measurable Condition-Set** template, presented in Figure 3, describes the basic measurable

units [32] of measurable condition sets, including temperature, length, charge, time, mass, angle, and luminous intensity. Any of these measurable specializations can override the basic **Condition Set** or **Invariant Set**. Non-basic measurable conditions can be modeled via the **Measured Condition Set** or by adding the relevant measurable unit to the template.

IV Guidelines for Eliciting Exceptions and incorporating them into a System's Conceptual Model via the Exception Templates

We suggest a methodology for conceptual modeling of complex, reactive, and safety-critical systems that supports exception specification as part of the design of the system's model**.** The first part of the methodology, which is usually done during system requirements or risk analysis, includes a set of guidelines for eliciting various ways in which the system can fail and then anticipating these situations by extending the basic "ideal" model. For all the possible values of each exception dimension, Table 2 suggests questions aimed at helping a system analyst or a knowledge engineer to think about all the failure modes of each normal-course process. These exception elicitation guidelines are independent of any system modeling method.

As demonstrated in the case study described in the next section, a system analyst could use this elicitation table (Table 2) as a guideline while reviewing all the system's normal processes in the system model, including tasks, blocks of tasks, and the global system process, in an iterative way. After completing the elicitation phase, the system analyst would eliminate redundant exceptions and exceptions that are considered out of scope or irrelevant.

Some dimensions of exception are not utilized in the exception elicitation and modeling guidelines. The Effect Scope, Predictability, and Severity dimensions are not used because they concern exception handling mechanism, including handling of non-predictable exceptions, which

were outside the scope of our work. The dimension of complexity of guarding conditions is covered elsewhere [29].

Having identified a list of possible exceptions, the second part of our methodology comprises a set of guidelines, presented in Table 3, for incorporating the elicited exceptions into the system's OPM conceptual model using the exception templates. The "Question for determining the value of the dimension" column should help the designer classify the various exceptions by their dimensional values into appropriate exception templates and their refinement through OPM, specified in the "Modeling Guidelines" column.

When modeling exceptions, system designers should traverse Table 3 top-down, using it as a decision tree, iteratively refining the selected exception template. As noted in Section III, the final refinement should incorporate resumption to normal execution (if applicable) via exception resolution. Modeling of exception resolution is beyond the scope of this paper. Nevertheless, exception resolution can be handled through regular OPM constructs.

We introduced the exception templates into the OPCAT [29] tool as meta-library templates and roles. This way, system architects can incorporate them into their OPM system model by tagging the system's objects and processes as performing roles of objects and processes of the meta-library, and then customize them for the specific system under development, as demonstrated in the next section.

V Modeling the Amoxicillin Antibiotics Treatment Clinical Guideline

In order to test the functionality and adequacy of the extended OPM framework and the modeling templates for specifying complex safety-critical systems with their exception-handling capabilities, we modeled several case studies, including auto-redial managing in a cellular phone [34], adapted from an actual industrial specification, and two clinical guidelines. One clinical

guideline was management of diabetic foot infection [29]. In this paper we present part of the model for the second clinical guideline—physician's consultation and outpatient treatment with Amoxicillin, adapted from instructions for patients on the use of that drug that were attached to the medication's package. The objective of modeling this example was to provide a model-based specification of the instructions for taking the medication, including different relevant exceptions that may arise and how they should be handled when detected or reported by the patient. In the normal process, once the physician has prescribed the antibiotics, the patient acquires the drug from the pharmacist and takes it according to the dosage prescribed. The dosage information specifies how many capsules of a certain weight of antibiotic substance the patient has to take each time, the frequency of taking the drug, and the period of time during which the drug has to be taken (e.g., take 1 capsule, every 8 hours, for 10 days).

The exception elicitation guidelines can be used to elicit exceptional situations. For example, considering the first item in the elicitation guidelines, goal violations, the recommended question directs the modeler to think about exceptions that would violate the goal of the system or would result in a need to change the goals of the system. The goal of the antibiotics treatment is to treat the patient for infection. An exception that may result in changing the goal could be a severe side effect that must be treated immediately. Hence, the immediate goal would be to stop the current treatment and treat the side effect.

Considering the fourth item in the elicitation guidelines, measurability, we are driven to think about exceptions that have to do with malfunctions of products stored in temperatures that are too high or too low, components that have predefined time limits, or limits on length, mass, luminous intensity, or charge. In the context of the antibiotics guideline, these considerations can help us to think about exceptions that are related to the drug: storing the drug at high temperature

(temperature violation), forgetting to take a dose in time (time violation), or taking too many pills at once (mass violation). In what follows, we show how the exception modeling guidelines (Table 3) are used to model the four exceptions described above. Figure 8 presents the **Antibiotics Acquisition and Treatment** process and demonstrates the utilization of the **Halting Invariants Set Violation** and **Parallel Violation** exception templates, which are surrounded by the dotted lines, representing the exceptions related to having side effects and storing the drug at a high temperature. The **Halting Invariants Set Violation** template, shown on the left-hand side of Figure 8, describes the event of developing drug side effects (denoted by entering the **ill with side effects** state of **Patient**), which can be triggered during the normal **Antibiotics Treatment** process. The "**developing side effect**" exception can be classified (when traversing Table 3) as asynchronous (as it does not happen at specific stages in the antibiotics treatment process), invariants condition violation, immeasurable, internal, of human cause, block specific in *Detection Scope* (since it can be triggered in any task that is part of its main process, **Antibiotics Treatment**), and rare. According to Table 3, the **Halting Violation Template** would be chosen for modeling this exception. When such an exception occurs, it halts the **Antibiotics Treatment** process and invokes the **Doctor Consultation** exception-handling process. Note that the template has been applied by tagging the **Antibiotics Treatment** as **Normal Processing**, the **Doctor Consultation** as **Exception Handling** and the **Patient** object as an **Invariant Set**.

The **Parallel Violation Template**, shown on the right-hand side of Figure 8, reflects an exception related to the violation of the guideline recommendation "Store the capsules and tablets at room temperature". Traversing Table 3, the storage temperature exception can be classified as asynchronous, invariants condition violation, measurable (temperature), external, non-human cause, block-specific in *Detection Scope* (since it can be triggered in any task that is

part of its main process, **Antibiotics Treating**), and rare. According to our modeling guidelines (Table 3), this exception is modeled using the **Parallel Invariants Violation template.** The **Storage Temperature** object is the **Invariants Set** for this template. The **Invariant Set** has been customized by overriding it with the **Measured Temperature Condition Set** structural template. The **normal values** were set to **24-30 Degrees Celsius** and **exceptional values** are **other** temperature values. If the normal values of the **Storage Temperature** are violated, i.e., the antibiotic is not stored at the correct temperature, then the **Antibiotics Replacement** process is triggered, as denoted by the event link (symbolized by the letter 'e' next to the event link) emanating from the **other** state of the **Storage Temperature** object and executed in parallel with the **Antibiotics Treatment** process. The **Antibiotics Treatment** process is executed in parallel with the **Antibiotics Replacement** process. This way, the antibiotics treatment is continued with the replaced drug that has not been stored in temperature values that are out of range. This is specified by the condition link (Instrument link with the letter 'c' inside the link's circle-head) emanating from the **24-30 Degrees Celsius** state to the **Antibiotics Treatment** process. Note that the **Antibiotics Treatment** process has been tagged with the exception meta-library role of **Normal Processing**, the **Antibiotics Replacement** process was tagged with the **Exception Handling** role, and the **Storage Temperature** object has been tagged with the **Measured Temperature Condition Set** role.

Figure 9 is an Object-Process Diagram (OPD) in which the **Antibiotics Treatment** process is zoomed into. It describes the procedure of the antibiotics home treatment after the **Patient** had been examined and the drug was prescribed by the doctor. The exceptions elicited above relate to problems with taking the drug that happen during the periodic process of **Taking Antibiotics**. In

order to follow the normal **Antibiotics Treatment** process and make sure that related exceptions do not occur, we need to have information regarding the dose taken and the scheduled times for taking the doses, and model this information. We first describe the modeling of the normal **Taking Antibiotics** process.

Taking Antibiotics, tagged as **Normal Processing**, produces a **Taken Dosage** object, tagged as a **Post Condition** of the process. The **Taken Dosage** object records the state of the dose taken. The dose taken during the process can be **none**, when the patient forgot to take the dose, **prescribed**, or **overdose**,

The **Taking Antibiotics** process is a periodic process that must occur with a specific period, as specified in the instructions for taking the drug. Therefore, the **Taking Antibiotics** process is triggered in the model by the **Timer** object**,** denoted by the event link emanating from it, which provides a triggering event every predefined time interval, as prescribed by the physician. **Timer** is a virtual object that reflects a specified time for taking the drug. It is used to monitor the time at which the antibiotics doses are taken and to calculate whether a missed dose could be taken later than prescribed, as explained below.

The normal process, **Taking Antibiotics,** is executed only if the following conditions are fulfilled: (1) there are capsules left (i.e., **capsules left > 0**) and (2) the medicine was stored at room temperature (**Storage Temperature** at state **24-30 Degrees Celsius**). After an instance of the **Taking Antibiotics** process has completed, the **Reset Timer** process (tagged as **Normal Processing**) resets the **Timer**, which then triggers the **Taking Antibiotics** process within another **Interval** of hours, thereby representing a periodic process.

According to Table 3, taking an overdose (more than the prescribed number of pills at a time) can be classified as synchronous (as it is detected after **Taking Antibiotics** is triggered by

Timer), post-condition violation, measurable (mass), external, human caused, task specific in **Detection Scope**, and rare. Therefore, as shown in the OPD of Figure 9, we use the **Post-condition Set Violation Template** to represent the exception related to overdose. If an **overdose** is taken (the **Taken Dose** object enters the **overdose** state), the patient needs to be referred to a hospital for further treatment, as specified by the triggering of the **Hospital Referring** process, tagged as **Exception Handling**.

We use Table 3 to classify the instructions regarding the fourth exception—realizing that the dose was not taken on time. Based on Table 3, we classify this exception as synchronous (since in our model it is detected after **Taking Antibiotics** is triggered by **Timer**), post-condition violation, measurable (time), internal, human caused, task specific in *Detection Scope*, and frequent. Therefore, as shown in the OPD of Figure 9, we use the **Post-condition Set Violation** template again to represent the exception related to a missed dose. This time, a post-condition of **Taken Dosage** being at the state of **none** (no dose taken) is a post-condition exception. It triggers two processes: the **Exception Handling** process **Taking Antibiotics Late** and the **Reset Timer** process, used to reset the timer so that it would signal the time to take the following dose. The **Taking Antibiotics Late** process enables the patient to take the antibiotics as late as one hour before the next dose, as specified by the event link emanating from the **none** state of **Taken Dosage** to the **Taking Antibiotics Late** process. This event link is tagged with the temporal constraint (0 h, Interval-1h), which specifies that this event will serve as a trigger only if it happens within the maximum time of 1 hour before the (periodic) **Interval** of taking the drug.

VII Discussion

Appropriate handling of exceptions is a key success factor in systems design and risk analysis. It is highly desirable to specify system exceptions and the mechanism for handling them early on

during the modeling phase of the system. When exceptions are not handled, or if they are handled inappropriately, the overall system's behavior becomes unpredictable and potentially hazardous. Our research has investigated the nature and characteristics of exceptions that can be envisioned and consequently modeled and taken care of during system design. We proposed a conceptual model and a framework for classification of exceptions that comprises extensive, domain-independent set of exception types and their characteristics. System analysts and knowledge engineers can use the exception elicitation guidelines, which are based on our conceptual model, to envision and characterize a large variety of exception types, independently of the modeling methodology used to represent the system and/or the exceptions. We extended Object-Process Methodology (OPM) to support modeling the different exception classes as standardized OPM design templates. The templates can support systems analysts to systematically analyze normal-course process models, anticipate possible exceptions, and provide a basis for augmented models, in which the normal-course scenarios are extended to include exceptions, their detection, and ways to handle them. The exception modeling templates can be utilized while creating these extended system models as standardized, predefined meta-model templates in OPCAT [9], which are imported as needed. In this paper, we demonstrated a specific application of our methodology for eliciting and modeling exceptions in a clinical care system.

To evaluate our exceptions elicitation methodology, we conducted a preliminary controlled experiment [29]. Undergraduate students in an Enterprise Systems Modeling course at the Technion – Israel Institute of Technology, were given a short textual description of the insert from a package of antibiotics. They were then asked to write down as many exceptions as they could think of. One group of students was given an earlier version of our exception elicitation

guidelines while the other group did not get the guidelines. The results showed that the group who was presented with the elicitation guidelines elicited significantly more exceptions in two categories: the goal of the process (the first dimension in the elicitation guidelines, presented in Table 3) and violation of measurable quantities (the fourth dimension in Table 3). Interestingly, although we presented only four exceptions in our case study example in Section V, the students came up with a total of 28 exceptions, averaging 7 exceptions per student.

Limitations and future work

While our paper addresses important aspects of exception handling, it does not solve all the problems in conceiving and modeling exceptions. First, we do not address unpredictable exceptions, which are very difficult to conceive. Second, any reasonable clinical guideline might have numerous exceptions. Hence, while it is possible to cover all the types of exceptions, it is impractical to explicitly enumerate each individual exception. Third, exceptions in clinical guidelines are likely to be more frequent and less predictable than exceptions in non-medical man-made systems that comprise software and hardware. While the designer of an artificial system normally has complete knowledge regarding the functionality and architecture of the system, clinical guidelines involve the human body—a highly complex natural and therefore much less predictable system. Moreover, clinical guidelines are expected to be followed by human users, including patients and clinical personnel, who are autonomous and capable of exercising flawed judgment. The patient directions used in this study are much more precise than recommendations found in clinical guidelines, which leave more room for flexibility and often contain ambiguous statements that leave room for clinical judgment. Thus, modeling the complete class of exceptions for clinical guidelines would in practice be more difficult than the example given in the Amoxicillin study.

Currently, our work does not address in detail the different types of exception handling process. Although exception handling strategies may be highly domain-specific, commonalities do exist, providing a basis for defining guidelines for eliciting exception handling strategies and patterns for their specification. Examples of exception handling processes from non-clinical domains include suspending a case or task, starting a new case, and sending warning messages to selected agents [14]. In the medical domain, exception handling processes include starting, stopping, aborting a therapy, postponing a therapy part, changing therapy properties, substituting therapy, and adding or deleting therapy step [11]. We plan to investigate the expressiveness of OPM for specifying exception handling recovery mechanisms and resumption to normal execution mode, including handling of unexpected events.

The exceptions classification and the exceptions elicitation guidelines can be employed in modeling methodologies other than OPM. We plan to study how the exception templates can be incorporated into specific clinical guideline/care flow formalisms, such as GLIF3 [35]. These formalisms have constructs that are specific to clinical workflows, such as patient information model, support for clinical decisions, and expression of decision criteria and clinical terms using controlled vocabularies. However, they lack adequate support of exceptions elicitation and modeling. Finally, we plan to conduct a thorough evaluation of the utility of our exception elicitation guidelines and modeling templates and their impact on the quality of the system's conceptual mode, created by system analysts.

VIII CONCLUSION

Our study demonstrated that OPM with its exceptions handling extension and modeling templates can be used to elicit and model exceptions in medical care systems. We modeled other systems, both from the medical domain – a clinical guideline for diabetic foot infections [29] –

and non-medical domains– a cellular phone redialing mechanism [34]. These diversified applications indicate that our methodology can likely be applied in a variety of domains, including medical patient care systems and real-time systems. Further research should be conducted to evaluate whether our methodology improves exceptions handling after it had been implemented in an actual real-life medical system.

ACKNOWLEDGEMENTS

We thank Samson Tu for his suggestions for improving the paper. We thank Yaron Denekamp, MD and Herbert Kaizer, MD for their help in checking the validity of the exception types found in the antibiotics case study, and we thank Dr. Kaizer also for validating the OPM model of the antibiotics case study.

References

1. Eberlein APG. Requirements Acquisition and Specification for Telecommunication Services: University of Wales; 1997.
2. Eder J, Liebhart W. Contribution to Exception Handling in Workflow Management. Proc. EDBT Work-shop on Workflow Management Systems; Valencia, Spain; 1998.
3. Klein M, Dellarocas C. A Systematic Repository of Knowledge about Handling Exceptions in Business Processes. ASES Working Paper ASES-WP-2000-03. Cambridge, MA: Massachusetts Institute of Technology; 2000.
4. Sadiq SW, Orlowska ME. On Capturing Exceptions in Workflow Process Models. Proceedings of the 4th International Conference on Business Information Systems; 2000; Poznan, Poland; 2000.
5. Klein M, Dellarocas C. A Knowledge-based Approach to Handling Exceptions in Workflow Systems. Computer Supported Cooperative Work (CSCW) 2000;9(3-4):399-402.
6. Dori D. Object-Process Methodology - A Holistic Systems Paradigm. Berlin, Heidelberg, New York: Springer Verlag; 2002.
7. Peleg M, Dori D. The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. IEEE Transactions on Software Engineering 2000;26(8):742-759.
8. Dori D. Syntax and Semantics of the Object-Process Methodology (OPM) Language. http://www.plosone.org/article/fetchFirstRepresentation.action?uri=info:doi/10.1371/journal.pone.0000872.s001, last access date 12/10/2008
9. Dori D, Reinhartz-Berger I, Sturm A. OPCAT - A Bimodal Case Tool for Object-Process Based System Development. 5th Intl Conf Enterprise Information Systems; 2003. p. 286-91.
10. Shlezinger G, Reinhartz-Berger I, Dori D. Analyzing Object-Oriented Design Patterns from an Object-Process Viewpoint. 3rd Intl Conf of Next Generation Information Technologies; 2006.
11. Greiner U, Mueller R, Rahm E, Ramsch J, Heller B, Loeffler M. AdaptFlow: protocol-based medical treatment using adaptive workflows. Methods Inf Med 2005;44(1):80-8.

12. Luo Z, Seth A, Kochut K, Miller J. Exception Handling in Workflow Systems. Applied Intelligence 2000;13:125-147.
13. Casati F, Ceri S, Paraboschi S, Pozzi G. Specification and Implementation of Exceptions in Workflow Management systems. ACM Transactions on Database Systems 1999;24(3):405-451.
14. Grefen P, Pernici B, Sanchez G. Database support for Workflow Management: the WIDE Project: Kluwer Academic Publishers; 1999.
15. Bock C. UML 2 Activity and Action Models, Part 6: Structured Activities. Journal of Object Technology 2005;4:43-66.
16. Russel N, van der Aalst WMP, ter Hofstede A. Exception Handling Patterns in Process-Aware Information Systems; 2006. Report No.: BPM Center Report BPM 06-04, BPMcenter.org.
17. Hagen C, Alonso G. Flexible exception handling in the OPERA process support system. 18th Intl Conf on Distributed Computing Systems; 1998.
18. Casati F, Pozzi G. Modeling Exceptional Behavior in Commercial Workflow Management Systems. 4th Intl. Conf. on Cooperative Information Systems; 1999.
19. Mulyar N, Pesic M, van der Aalst, WMP, Peleg M. Towards Flexibility in Clinical Guideline Modelling Languages. Business Process Modeling Conference Workshop: 1st International Workshop on Process-oriented Information Systems in Healthcare; Brisbane, Australia; 2007.
20. Chiu DKW, Li Q, Karlapalem K. ADOME-WFMS: toward cooperative handling of workflow exceptions. In: Advances in Exception Handling Techniques; 2001. p. 271-88.
21. Chiu DKW, Li Q, Karlapalem K. Exception Handling with Workflow Evolution in ADOME-WFMS: a taxonomy and resolution Techniques. ACM SIGGROUP Bulletin 1999;20:8.
22. Hwang SY, Ho SF, Tang J. Mining exception instances to facilitate workflow exception handling. 6th Intl Conf on Database Systems for Advanced Applications; 1999.
23. Luo Z, Sheth A, Kochut K, Arpinar B. 2002. Exception Handling for Conflict Resolution in Cross-Organizational Workflows: LSDIS Lab, Computer Science, Uni. of Georgia; 2002.
24. Eder J, Liebhart W. The Work-flow Activity Model WAMO. 3rd Intl Conf on Cooperative

Information Systems; 1995.

25. Fridsma DB, Thomsen J. Representing Medical Protocols for Organizational Simulation: An Information-Processing Approach. Computational & Mathematical Organization Theory 1998;4(1):71-95.
26. Quaglini S, Stefanelli M, Lanzola G, Caporusso V, Panzarasa S. Flexible Guideline-based Patient Careflow Systems. Artificial Intelligence in Medicine 2001;22:65-80.
27. Brambilla M, Ceri S, Comai S, Tziviskou C. Exception handling in workflow-driven Web applications. 14th Intl Conf on WWW; 2005.
28. Avizienis A, Laprie JC, Randell B. Fundamental Concepts of Dependability: UCLA CSD report #010028; 2000.
29. Somekh Y. Modeling Exceptions with Object Process Methodology: M.Sc Thesis, Technion - Israel Institute of Technology; 2007.
30. Casati F, Cugola G. Error Handling in Process Support Systems. In: Advances in Exception Handling Techniques: Springer; 2001. p. 251-270.
31. Sutton M. Preconditions, Postconditions and Provisional Execution in Software Processes: LASER Technical Report 95-77. University of Massachusetts, Amherst.; 1995.
32. Schadow G, McDonald CJ, Suico JG, Fohring U, Tolxdorff T. Units of Measure in Clinical Information Systems. J Am Med Inform Assoc 1999;6(2):151-161.
33. Kim MH, Park YM, Yang SM, Park JK. Modeling of a Highly Reliable Real-Time Distributed System using the RTO.k Model and the Monitor Object. Proceedings of the 3rd Workshop on Object-Oriented Real-Time Dependable Systems, p. 48, 1997.
34. Somekh Y, Peleg M, Dori D. Classifying and Modeling Exceptions through Object Process Methodology. Proc. International Conference on Systems Engineering and Modeling 2007.
35. Boxwala AA, Peleg M, Tu S, Ogunyemi O, Zeng Q, Wang D, Patel VL, Greenes RA, Shortliffe EH. GLIF3: a representation format for sharable computer-interpretable clinical practice guidelines. J Biomed Inform 2004;37(3):147-61.

Figure Legends

Fig. 1. OPD and OPL of the exception conceptual model

Fig. 2. An OPD showing the attributes of Triggering Object

Fig. 3. Measurable Condition Set Template

Fig. 4. Zooming into Exception Handling process in the OPM-based exception meta-model.

Fig. 5(a) Precondition Set Violation Template; (b) Postcondition Set Violation Template

Fig. 6 Halting Invariants Set Violation Templates - Exception Handling halts Normal Processing execution. (a) Explicit Invariant Set Violation Template – the Invariant Set is modeled explicitely; (b) Event Invariants Set Violation Template - Exception Handling results from triggering an event. The Invariant Set is implicit

Fig. 7. Parallel Invariants Set Violation Template - Exception Handling is executed in parallel to Normal Processing

Fig. 8. Object-Process Diagram (OPD) of the Antibiotics Acquisition and Treating process in-zoomed

Fig. 9. OPD of the Antibiotics Treating process in-zoomed

Table 1. Dimensions of exceptions defined by different research works

Research	Domain	Detection Scope	Effect Scope	Synchronicity	Constraint Violation	Internal / External (Origin)	Predictability	Measurability	User Errors (essence, cause)	Frequency	Severity	Exception Handling
Fridsma and Thomsen [25]	medical care								+			
Quaglini et al. [26]	medical care		+	+			+					
Casati and Pozzi [18]	workflow systems		+	+								
Brambilla et al. [27]	Workflow-driven Web applications	+	+	+					+			
Luo et al. [5]	cross-organizational workflow systems											+
Eder and Liebhart [2,6]	WAMO – Workflow Activity Model	+					+					+
Russel et al. [7]	Process-aware Information Systems	+			+	+		+ Time exceptions (deadline expiration)				
Avizienis et al. [8]	Computing Systems	+				+		+ Time	+		+	
Sadiq and Orlowska [4]	Workflow Systems	+	+				+			+		+
Klein and Dellarocas [5]	Workflow Systems	+							+			+
Chiu and Karlapalem [21]	Workflow Systems					+						

+ indicates that the dimension is covered in the reference paper

Table 2. Guidelines for eliciting exceptions

Characteristic / Dimension	Dimension value	Elicitation Question
Goals violation and Detection Scope	Task /Block /System	Think of the process' goals, considering its optimal execution. Can some of these goals be violated by some occurrence? Can some of these goals be changed by some occurrence?
Synchronicity	Asynchronous (Event)	Think of all the events that can occur at any point or any time during the process execution. What exceptions can result from them?
	Synchronous (Branch)	Think of all the decision points during the process flow. Are there paths that are considered less optimal and can be considered as exceptional?
Condition	Preconditions	Think of all the assumptions and conditions that must be true for the process to start execution. What exceptions result from the violation of these assumptions and conditions?
	Invariants	Think of all the assumptions and conditions that must be true during the process execution. What exceptions result from the violation of assumptions and conditions that must be enforced during process execution?
	Postconditions	Think of all the assumptions and conditions that must be true for the process to terminate successfully. What exceptions result from the violation of the assumptions and conditions that restricts the process successful termination?
Measurability	Measurable	Does the process utilize components (e.g., products) that may malfunction if they are not kept in a certain temperature range? This can cause an exception. Does the process utilize components (e.g., products) that should function in predefined time limits? Age limit? Similarly, is there a required range for length, charge, mass, angle luminous intensity or for other measurable units related to components participating in the process? Are there patient data items that are considered exceptional if they are outside some range?
	Immeasurable	Think of unwanted actions and situation that are not constrained by time, temperature or other measurable values. What exceptions are related to them?
Origin	Systemic	Think of all the internal objects participating in the process (e.g., internal database, tasks etc). What exceptions can they trigger? What exceptions can be related to them?
	Environmental	Think of all the external objects participating in the process (e.g., equipment, external organization etc). What exceptions can they trigger?
Essence and Human Cause	Human	Think of all the humans participating in the process. What exceptions can they trigger? Consider errors, malicious action, non-compliance, and suboptimal coordination or communication What exceptions can be related to humans but due to unexplained malfunction that is not the fault of the human? (e.g., allergic reaction, a new symptom, a finding, side effects)
	Non-human	Think of all the non-human objects participating in the process. What exceptions can they trigger? Consider organizational causes (e.g., resource unavailability), work item failure (e.g., problem with a laboratory test), deadline expiration, hardware/software incompatibility.
Frequency	Frequent	Think of the frequent and rare behaviors that deviate from the ideal process' flow. What exceptions are related to them?

Table 3. Guidelines for modeling exceptions in OPM using design templates

Step	Dimension	Question for determining the value of the dimension	Answer and determined Dimension value	Modeling Guidelines
1	Synchronicity	Can the exception occur at any point during process execution?	Yes=Asynchronous (Event)	Use **Invariant violation templates**. Use one of the halting templates if the initial process is halted or the parallel template otherwise.
			No=Synchronous (Branch)	Use **Pre/Post-condition violation templates**
2	Condition	Does the exception results from the violation of assumptions and conditions that must be true prior to the process execution? If no, proceed to the next question.	Yes=Preconditions violation	Use **Precondition violation template**
		Does the exception results from the violation of assumptions and conditions that must be enforced during process execution? If no, proceed to the next question.	Yes=Invariants violation	Use **Invariant violation templates**
		Does the exception results from the violation of assumptions and conditions that restricts the process successful termination?	Yes=Postconditions violation	Use **Postcondition violation template**
3	Measurability	Does the exception result from the violations of quantified measures?	Yes = Measurable	You may use the **Measurable Condition Set template** for overriding the Condition set
			No = Immeasurable	If it is compound condition set (contains logical statement(s)), and logics encapsulation is needed: zoom into **Condition Set** as presented in the **Compound Condition Set template**.
4	Origin	Does the exception triggered by internal system objects or components?	Yes=Internal	The triggering entity should be specified in OPM as a systemic object
			No=External	The triggering entity should be specified in OPM as an environmental object
5	Essence and Human Cause	Does the exception results from a human misconduct?	Yes=Human	Use physical object for the triggering entity Use agent link for connecting the triggering object to the process
			No=Non-human	Use instrument link for connecting the triggering object to the process
6	Detection Scope	Does the exception related to the system globally such that it can occur in each of the system's tasks? If no, proceed to the next question.	Yes=System	Exception-handling process should be located in the top levels of the OPD tree (in the System Diagram or one level below it). It can be refined or overridden in lower leveled OPDs.
		Does the exception can occur and is related to some block of tasks? If no, proceed to the next question.	Yes=Block	Exception-handling process should be located in the relevant OPD that includes the block (presented as a general process) or the relevant tasks (consisted in the block). It can be refined in lower leveled OPDs.
		Does the exception can occur and is related only one specific task?	Yes=Task	Exception-handling process should be located in the OPD that includes the specific atomic task description.
7	Frequency	Does the exception occur frequently?	Yes=Frequent	Usually, exception-handling process should to be part of the normal OPD and embedded in it [4]
			No=Rare	Exception-handling process should be modeled usually in a separate OPD.