Classifying and Modeling Exceptions through Object Process Methodology

Yudit Somekh
Faculty of Industrial Engineering and Management
Technion – Israel Institute of Technology, Haifa, 32000, Israel
yuditol@tx.technion.ac.il

Mor Peleg
Department of Management Information Systems,
University of Haifa, Haifa 31905, Israel
morpeleg@mis.hevra.haifa.co.il

Dov Dori
Faculty of Industrial Engineering and Management
Technion – Israel Institute of Technology, Haifa, 32000, Israel
dori@ie.technion.ac.il

Abstract

Exception handling is a fundamental issue that needs to be addressed by complex systems such as embedded systems, real time systems and medical information systems. Modeling exceptions appropriately is a crucial step in a system design process, since correcting errors detected after the design phase can be very costly. Object Process Methodology (OPM) is a methodology that uses a single graphical model for describing systems, including their timing exceptions, and has been shown to be an effective modeling methodology. We developed a taxonomy of exceptions, and refined and extended OPM for expressing the different types of exceptions and their detection and handling. We exemplify the introduced concepts using a case study of a cellular phone, taken from a real industrial system.

1. Introduction

Modern systems are becoming ever more complex, increasing the number of exceptional situations they have to cope with. There is a shortage of methods for supporting system architects and designers with methodologies that allow them to model and handle exceptions correctly. Exception handling mechanisms that are specific to various application domains and design paradigms are also scarce.

Software errors detected after the system's design phase ends account for significant software development costs if their resolution requires system redesign [1]. For example 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 [2]. There is a tremendous potential to save such costs and time through improving the requirements and modeling practice, especially via exceptions managing. This can be achieved by developing and adopting methods that model a wide range of exceptions during the design phase of a system's lifecycle.

There are many definitions for exceptions [3-5]. Influenced by the definition given in [3] that relates to a process' goal, we define an exception as an occurrence which deviated from the ideal normal flow necessitating a change in the primary goal of the current task, which may lead to change of important goals of the entire system.

Other researchers have tried to model exceptions, either through embedded approaches that embeds the exceptional semantics within the model of normal system behavior, or through stand-alone approaches that separates the exceptional semantics from this model [4, 6-9] Most of the stand-alone approaches adopted ECA rules as modeling tools [10-15].

Our work is focused on supporting system designers with a design methodology for modeling abnormal behaviors and exceptions that can be predicted in advance and developing mechanisms for handling them during the design phase.

The design of exceptions and their handling mechanisms requires understanding the nature of the wide range of potential exceptions as well as the ability to represent them by using explicit modeling constructs

in a solid methodology. Object-Process Methodology (OPM) [25], a graphical and textual modeling methodology that includes the behavioral and structural aspects of a system in a single model, was chosen as the framework to model exception handling. OPM was shown experimentally [16] to be effective in producing system specifications of high quality, compared to OMT – the main ancestor of the Unified Modeling Language (UML) – the industrial system analysis and design standard de-facto. OPM is suitable for modeling dynamic systems as it can directly express events, Event-Condition-Action (ECA) rules, guarding conditions[1] and timing exceptions [17]. However, it lacks the ability to model the full range of exceptional behaviors such as: asynchronous non-temporal exceptions, and has no means for representing uncertainties of the guarding conditions and for simplifying the specification of complex conditions.

The goals of the work presented in this paper are to support system designers with an augmented OPM methodology that would appropriately represent the possible exceptions that may occur and the actions that should be performed to handle them. In order to improve exception understanding and coverage, a classification and meta-model of exceptions is presented. The classification is exemplified using a cellular-phone case-study, taken from a real industrial system, which is analyzed using the augmented OPM.

2. Characteristics and Taxonomy of Exceptions

The first step of our work was to analyze the characteristics of exceptions and define a taxonomy that supports the system designer while modeling exceptions. Our analysis was performed by modeling real case-studies demonstrating complex dynamic environments taken from the medical-care area and the real-time area. The developed taxonomy is based on our analysis and the related literature [3, 6, 18-20].

This taxonomy should help the systems designer to better understand the nature of possible system exceptions and their management.

The top-level diagram of the exception meta-model is shown in Figure 1. An Exception is generated by Triggering Entities that can be external or internal to the system. Like the ECA paradigm, an Exception consists of a Trigger (or multiple Triggers) and optional Guarding Conditional Statements, and it is characterized by *Exception Handling* procedures. The Trigger and the Guarding Conditional Statements participate in the *Handling* process which is responsible for the management and resolution of the Exception.

Exceptions can be characterized and classified according to the following criteria:

Trigger Type – an *event* (i.e., a significant occurrence in the system at a specific time point) or a *branch* (i.e., a decision point) in the process. A path generated from some *branch* is considered to be exceptional if it is less favorable for meeting the process goal compared with other path(s) emanating from the same branch.

Predictability – an attribute denoting whether the exceptions can be foreseen at design time (*expected*) or not (*unexpected*).

Source – an exception's source can be classified as *internal* (systemic) failures of system components or *external* (environmental) failures [14]. External exceptions can be human- or non-human generated [19]. Human-generated exceptions can be caused by human errors, non-compliance, or malicious actions [22].

Synchronicity – exceptions may be *synchronous* (i.e., branches of abnormal behavior taken at pre-specified decision points) or *asynchronous* [6].

Frequency – exceptions can be *frequent* or *rare*; *frequent* exceptions tend to be part of the normal flow and are usually embedded in the model [4]. *Rare* exceptions will be modeled usually in a separated procedure.

Measurability – exceptions can be classified according to their ability to be measured. *Measurable* exceptions are triggered when a given measurable value is reached (or elapsed) or some limited boundary is violated. They are measured by measurement units, which are composed from the basic units of temperature, length, charge, time, mass, angle and luminous intensity [23], and can occur once or in a periodic manner. A particular example of *measurable* exceptions is temporal exceptions [6]. *Immeasurable* exception cannot be quantified by means of measurable units. An example for this kind of exception is failure of a network connection.

[1] Guarding conditions are pre-conditions that guard the execution of a process, which is executed if and only if its set of zero or more guarding conditions is met.

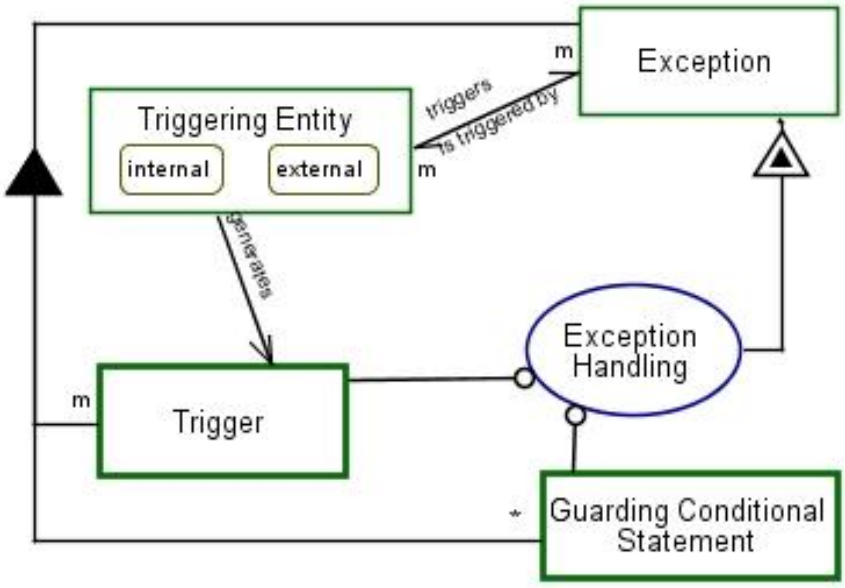


Figure 1. A top-level diagram of the OPM-based exception meta-model

Scope Level – the scope of a failure or an exceptional occurrence can be divided into three levels [4, 6]: *task-level* failure (a failure that occur and is related to one specific task), *block-level* failure (failure that is related to and can occur within some block of tasks), and *system-level* failure (failure that is related to the global system itself and can happen in each of its tasks). *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* exception.

We distinguish *detection scope* from *resolution scope*. As mentioned, detection scope concerns the points at which an exception can occur, while resolution scope concerns the effect of the exception or of the resolution process on current or future system processes.

Severity – the severity [24] of exceptions is determined by their potential effect on the normal operation of the systems, which can be *ignorable*, *light*, *true*, and *hard*. *Ignorable* exception does not affect the normal operation of the system, so it can be ignored and no treatment is necessary. *Light* exception does not cause any error but a continuous supervision of the faulty component is required and it may require the execution of a recovery function. *True* exception causes malfunction of main system components but there are replaceable components or other ways of recovery (that usually help in reaching the original goal) are possible. In a *hard* exception, 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 settings which usually results in the termination of the current task.

3. The Cellular Phone Case Study

The modeling language of a system model is required to adequately manage exceptions by means of offering a framework, composed of set of mechanisms, to react to the different classes of exceptions identified in Section 2.

OPM's extended framework and the developed taxonomy are illustrated and introduced in an example scenario of real-time cellular-phone managing. The focus is on the processing of automatic re-dialing initiated by some source cellular-phone to some destination cellular-phone, as described in the specification below.

3.1 Cellular Phone Automatic Redialing Specification

If enabled by the user, the phone shall re-dial unanswered numbers until the caller answers, or one of the termination conditions is reached. The complete specification of conditions for redialing attempts is given in Appendix A.

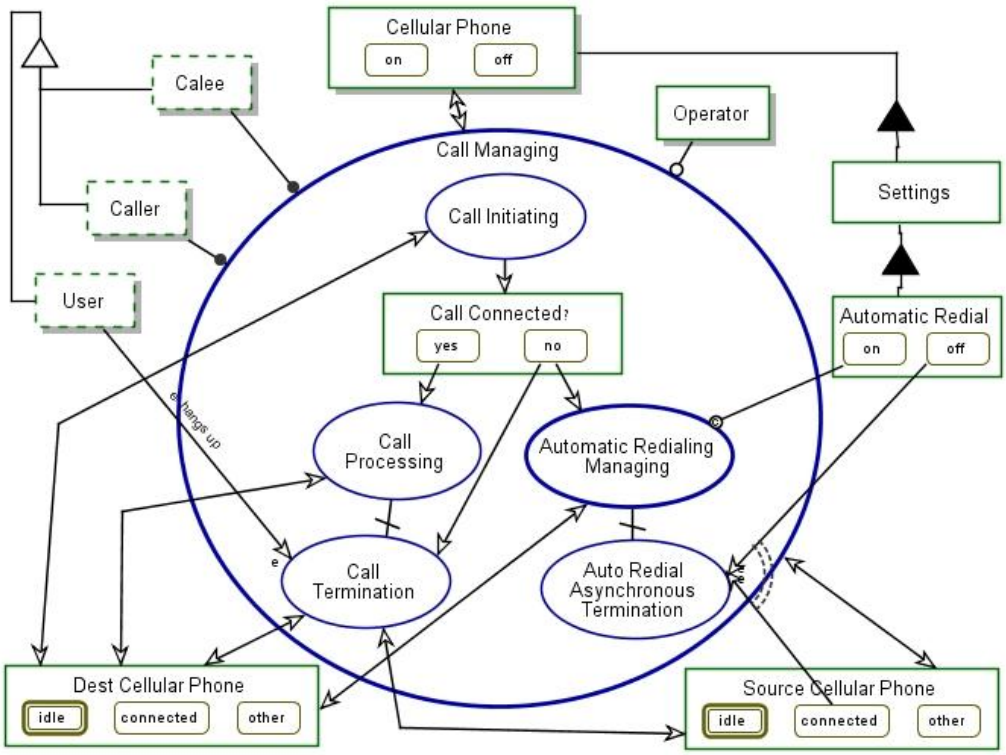


Figure 2. In-zooming of the *Call Managing* Process

3.2 Cellular Phone Automatic Redialing Specified through OPM

Figures 2 through 4 constitute the OPD- (Object Process Diagram) set that describes the specification of the process of cellular phone automatic redialing.

The *Call Managing* process is presented in Figure 2. This process includes *Call Initiating* process that concerns actions such as dialing and waiting for a response. If *Call Initiating* succeeds, a call is connected and processed; if it fails and the connection couldn't be established, a deviation from the ideal behavior occurs, thus exceptional behavior should be handled. The exceptional behavior is triggered by the following Trigger: Call Connected? enters the "no" state. This exception is characterized as synchronous *branch*. *Exception Handling* includes two processes that are executed in parallel: *Call Termination*, which generates termination actions concerning the current manual attempt, and a new *Automatic Redialing Managing* attempt that is initiated. The Guarding Conditional Statement is that Automatic Redial (of the Settings object) is enabled by the user (thus is in the "on" state).

In addition, the exceptional situation can be classified as *internal or external* failures *(depends on the failure type*, e.g. "network out of order" is an *external* failure and "line busy" is an *internal* failure*), immeasurable*, *task-specific* and of severity *'true'*. The Asynchronous exception handling process, *Auto Redial Asynchronous Termination*, is triggered by at least one of the

triggering entities - Source Cellular Phone entering the "connected" state (because a call had been accepted by the callee) or Automatic Radial settings entering the "off" state (disabled by the user). The asynchronous exception is expressed by an exception link from the main process to the exception handling process. The triggering events and guarding conditions, which ensure the exceptional handling, are connected to the exception handling process. The *Auto Redial Asynchronous Termination* exception handling process can be classified as *event, external, asynchronous, immeasurable, block specific* in detection (since it can be triggered in every task that is part of its main process, *Automatic Redialing*) and having severity of *true*. The handling scope is *block specific* as well, since stopping the auto redialing attempts affects the future auto-redialing operations.

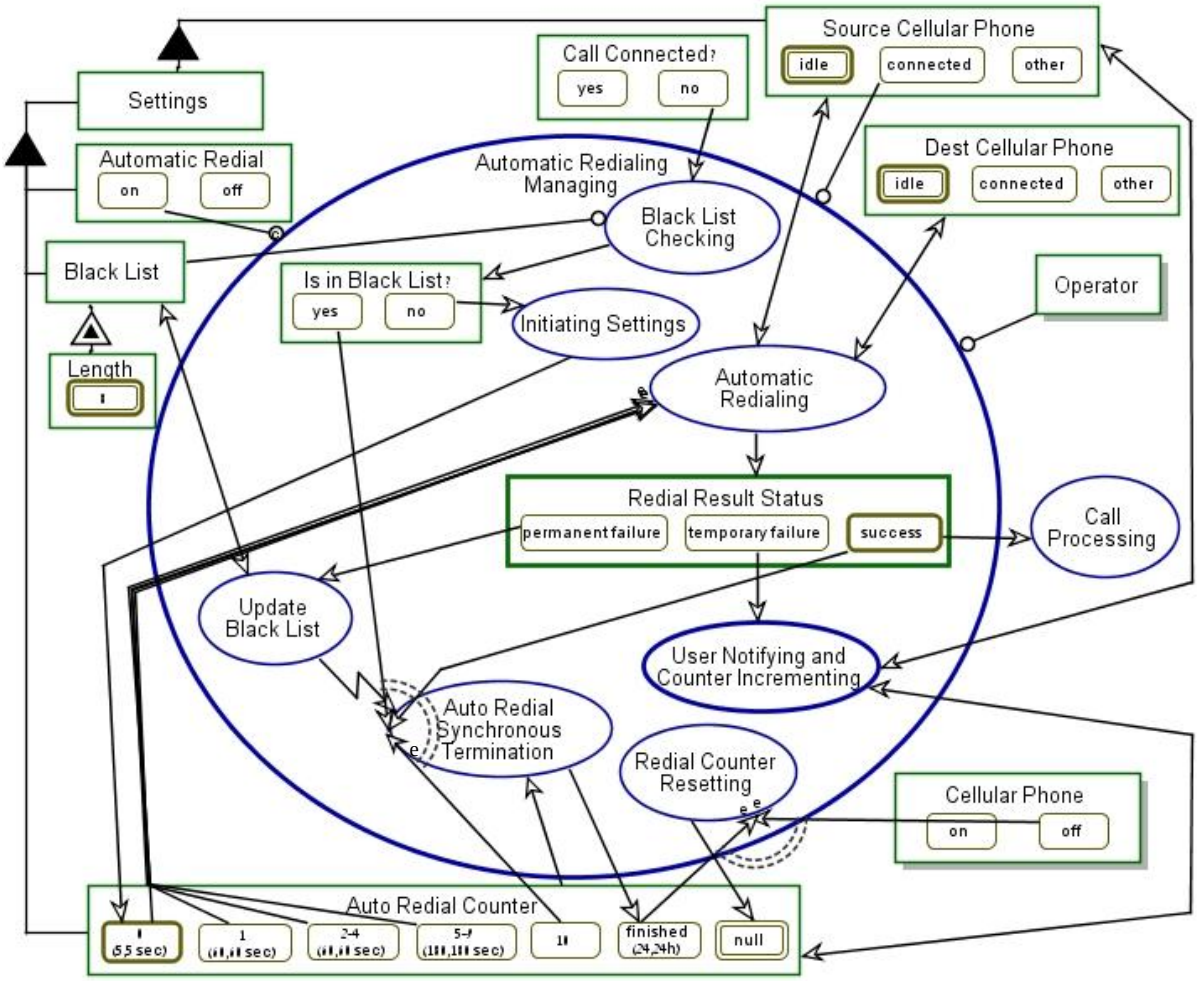


Figure 3 In-zooming of the *Automatic Redialing Managing* Process

The *Automatic Redialing Managing* process is zoomed into in Figure 3. After Checking that the called number is not in the black list, the Auto Redial Counter object is initiated, by entering its "0" state. This starts the automatic redialing execution by triggering the *Automatic Redialing* process after the duration of staying in the state has elapsed (5 seconds). The process results in the object Redial Result Status which is a decision point for the proceeding of the flow. If the execution of *Automatic Redialing* process results in the "permanent failure" state (of Redial Result Status Triggering Entity) the black list is updated and an *Auto Redial Synchronous Termination* is invoked. If this process results in the "temporary failure" state of the Redial Result Status Triggering Entity, the user is notified with the failure, the Auto Redial Counter is

incremented and the *Automatic Redialing* process is executed again, but only after the maximal duration of the counter's new state has elapsed (denoted by the event link connecting Auto Redial Counter relevant state to the *Automatic Redialing* process). If this temporary failure resulted after the 10^{th} attempt (meaning that the Triggering Entity - Auto Redial Counter entered its "10" state), the *Auto Redial Synchronous Termination* is invoked as well. If the Redialing attempt succeeded, meaning that a call was initiated, the *Call Processing* process is executed in parallel to the *Auto Redial Synchronous Termination* process. The Process *Auto Redial Synchronous Termination* transforms the state of the Auto Redial Counter to "finished". After the maximal duration of the "finished" state elapses (24 hours) or if the Cellular Phone state is changed to "off", the *Redial Counter Resetting* process is triggered and resets the Auto Redial Counter to its "null" state. The temporary and permanent failures of auto redialing attempts are both exceptional situations that can be classified as *branch, internal or external (depends on the failure type), synchronous, immeasurable, task-specific* in their detection and *block-specific* in their handling (since they affect future redialing attempts to the failed number), and of severity *true*. An abort operation is executed in the *Auto Redial Synchronous Termination* exceptional process.

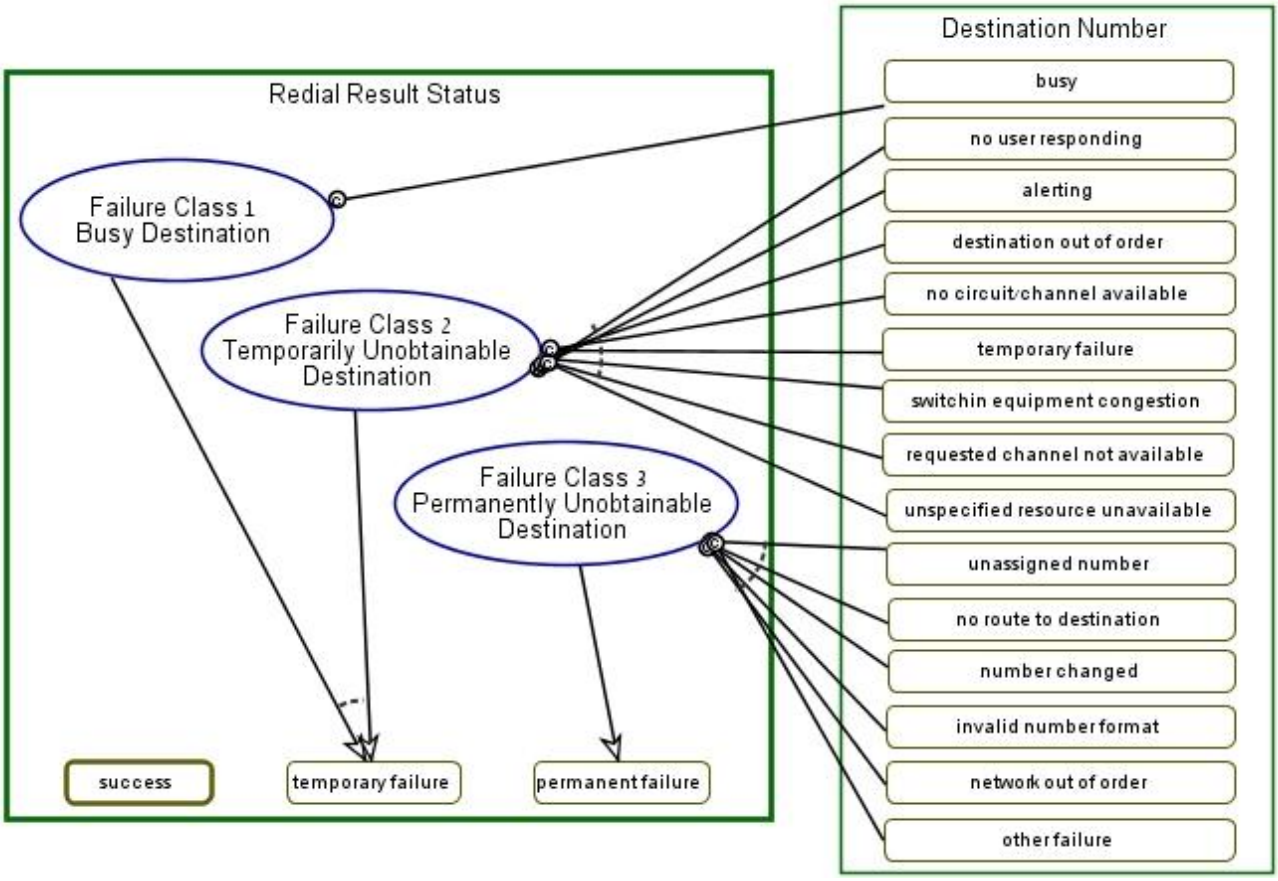


Figure 4 In-zooming of the Redial Result Status object

Guarding conditions of exceptional operations are frequently assembled into complex logical statements. This tends to make the model and its diagrams cluttered and complex. In order to be able to simplify the diagrams and their logical statements, we developed an encapsulation mechanism by using the in-zooming mechanism of OPM on decision constructs. To exemplify this mechanism, consider the object Redial Result Status which encapsulates the logics of the decisions about the failure types. The encapsulated logic, presented in Figure 4, is reached by zooming into the object Redial Result status. This object has only three abstract states (success, temporary failure, and permanent failure) that are

derived from 25 concrete states from the domain problem, as hereby described. One of the states of an OPM object is usually selected as an initial state; in the Redial Result Status object, the initial state is the "success" state that occurs if no failure happens. The temporary failure happens if the destination is busy or temporarily unobtainable. If the destination is permanently unobtainable, the "Redial Result Status" object resumes in its "permanent failure" state. The logical conditions are expressed with the condition links that can be manipulated with the and/xor/or operations [25]. In addition, the parts of a complex statement that share similar meaning are gathered together, thus simplifying the understanding and abstracting the logic. For example, the logical statements for temporarily unobtainable destination are gathered together, thus separating this logical part from the statement and improving the visual understanding of the whole statement. It should be noted that sometimes the logical statement cannot be evaluated since some values are uncertain, thus an "unknown" or "null" state are added to the decision object.

4. Discussion

Exceptions are a very important aspect of a system's behavior and it is very important to specify them 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.

Our research has investigated the nature and characteristics of exceptions that can be envisioned during system design. Its result is an ontology of exceptions that comprises an extensive, domain-independent classification of exception types and their characteristics. We have extended the Object-Process Methodology (OPM) to support the exceptions ontology. The OPM exception-handling extensions include support of *measurable* exceptions and general asynchronous exceptions. The utilization of objects and processes in OPM, along with its built-in complexity-management mechanisms, enabled us to develop simplifying shortcuts for describing complex guarding conditions with their possible uncertainties, using encapsulation and abstraction methods.

The expressiveness of OPM with its exception handling extensions has been evaluated with favorable results through several real industrial case studies, parts of which are presented in this paper, from the domains of real time systems and medical care flow systems that were enriched with almost all the possible exception types in our exceptions ontology.

We plan to extend the exceptions ontology to exception handlers as well as developing guidelines to support systems architects and designers in modeling abnormal behaviors and covering the various exception types. We plan to develop exception design patterns that can be utilized while creating system models as role constructs imported from OPM's meta model through the OPCAT modeling tool [26].

We plan to also further investigate the expressiveness of OPM for specifying exception handling, recovery mechanisms, and the resumption to normal execution mode. The notion of uncertain values of the conditional states should also be extended and formalized.

Finally, the ability of system developers to use our exceptions ontology and the methodology of figuring them out and modeling them effectively and easily should be investigated.

Acknowledgement

The partial support of this research by the Bernard M. Gordon Center for Systems Engineering is gratefully acknowledged.

References

[1] J. C. Westland, "The cost of errors in software development: evidence from industry," Journal of Systems and Software, vol. 62, pp. 1-9, 2002.

[2] A. P. G. Eberlein, "Requirements Acquisition and Specification for Telecommunication Services," PhD Thesis, University of Wales, November 1997, pp. 36-7.

[3] M. Klein and C. Dellarocas, "A Systematic Repository of Knowledge about Handling Exceptions in Business Processes," ASES Working Paper ASES-WP-2000-03, Massachusetts Institute of Technology, Cambridge, MA, USA August, 2000.

[4] S. W. Sadiq and M. E. Orlowska, "On Capturing Exceptions in Workflow Process Models," presented at Proceedings of the 4th International Conference on Business Information Systems, Poznan, Poland, April 12 -13 2000.

[5] J. Eder and W. Liebhart, "Contribution to Exception Handling in Workflow Management," presented at Proceeding of the EDBT Workshop

on Workflow Management Systems, Valencia, Spain,, 1998.
[6] F. Casati and G. Cugola., "Error Handling in Process Support Systems," in Advances in Exception Handling Techniques: Springer, 2001, pp. 251-270.
[7] J. Eder and W. Liebhart, "The Work-flow Activity Model WAMO," presented at Proc. E 3rd Int. Conference on Cooperative Information Systems, Vienna, Austria, 1995.
[8] F. Casati and G. Pozzi, "Modeling Exceptional Behavior in Commercial Workflow Management Systems," presented at 4th Intl. Conf. on Cooperative Information Systems, 1999.
[9] C. Bock, "UML 2 Activity and Action Models, Part 6: Structured Activities," Journal of Object Technology, vol. 4, pp. 43-66, 2005.
[10] Z. Luo, A. Seth, K. Kochut, and J. Miller, "Exception Handling in Workflow Systems," Applied Intelligence, vol. 13, pp. 125-147, 2000.
[11] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and Implementation of Exceptions in Workflow Management systems," ACM Transactions on Database Systems, vol. 24, pp. 405-451, 1999.
[12] P. Grefen, B. Pernici, and G. Sanchez, Database support for Workflow Management: the WIDE Project: Kluwer Academic Publishers, 1999.
[13] D. K. W. Chiu, Q. Li, and K. Karlapalem, "ADOME-WFMS: toward cooperative handling of workflow exceptions," in Advances in Exception Handling Techniques, 2001, pp. 271-288.
[14] D. K. W. Chiu, Q. Li, and K. Karlapalem, "Exception Handling with Workflow Evolution in ADOME-WFMS: a taxonomy and resolution Techniques," ACM SIGGROUP Bulletin, vol. 20, pp. 8, 1999.
[15] S. Y. Hwang, S. F. Ho, and J. Tang, "Mining exception instances to facilitate workflow exception handling," presented at Proceeding of the 6th International Conference on Database Systems for Advanced Applications, 1999.
[16] M. Peleg and D. Dori, "The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods," IEEE Transactions on Software Engineering, vol. 26, pp. 742-759, 2000.
[17] M. Peleg, "Modeling System Dynamics Through The Object-Process Methodology," PhD Thesis, Technion - Israel Institute of Technology. Subject No. 681.3.06 OOP, System No. 2204924, April 1999.
[18] S. Quaglini, M. Stefanelli, G. Lanzola, V. Caporusso, and S. Panzarasa, "Flexible Guideline-based Patient Careflow Systems," Artificial Intelligence in Medicine, vol. 22, pp. 65-80, 2001.
[19] M. Brambilla, S. Ceri, S. Comai, and C. Tziviskou, "Exception handling in workflow-driven Web applications," presented at Proceedings of the 14th international conference on World Wide Web, 2005.
[20] A. Avizienis, J. C. Laprie, and B. Randell, "Fundamental Concepts of Dependability," UCLA CSD report #010028 2000.
[21] M. Klein and C. Dellarocas, "A Knowledge-based Approach to Handling Exceptions in Workflow Systems," Computer Supported Cooperative Work (CSCW), vol. 9, pp. 399-402, 2000.
[22] D. B. Fridsma and J. Thomsen, "Representing Medical Protocols for Organizational Simulation: An Information-Processing Approach," Computational & Mathematical Organization Theory, vol. 4, pp. 71-95, 1998.
[23] G. Schadow, C. J. McDonald, J. G. Suico, U. Fohring, and T. Tolxdorff, "Units of Measure in Clinical Information Systems," J Am Med Inform Assoc, vol. 6, pp. 151-161, 1999.
[24] M. H. Kim, Y.-W. Park, S.-M. Yang, and J.-K. Park, "Modeling of a Highly Reliable Real-Time Distributed System Using the RTO.k Model and the Monitor Object," presented at Proceedings of the 3rd Workshop on Object-Oriented Real-Time Dependable Systems, 1997.
[25] D. Dori, Object-Process Methodology - A Holistic Systems Paradigm. Berlin, Heidelberg, New York: Springer Verlag, 2002.
[26] D. Dori, I. Reinhartz-Berger, and A. Sturm, "OPCAT - A Bimodal Case Tool for Object-Process Based System Development," presented at 5th International Conference on Enterprise Information Systems, 2003.

Appendix A: Cellular Phone Automatic Redialing Specification

If enabled by the user, the phone shall re-dial unanswered numbers until the caller answers, or one of the termination conditions is reached.

Redialling attempts shall be made as follows:

Call Attempts	Minimum Duration Between Attempts
Initial	N/A
1st repeat	5 sec.
2nd repeat	1 min.
3rd repeat	1 min.
4th repeat	1 min.
5th repeat	3 min.
…	…
nth repeat	3 min.

A redialling attempt shall be considered to have failed if it encounters one of the following conditions:

Class	Condition
Class 1 – busy destination	User busy
Class 2 – unobtainable destination (temporary)	No user responding
	User alerting, no answer
	Destination out of order
	No circuit/channel available
	Temporary failure
	Switching equipment congestion
	Requested circuit/channel not available
	Unspecified resources unavailable
Class 3 – unobtainable destination (permanent or long-term)	Unassigned number
	No route to destination
	Number changed
	Invalid number format (also incomplete number)
	Network out of order

Termination Conditions

- Maximum Number of Attempts

Up to N redialling attempts shall be made before assuming that the dialled number is unobtainable. For failure classes 1 and 2, N shall be 10. For failure class 3, N shall be 1. The count of redialling attempts shall be maintained for 24 hours, or until the phone is switched off, whichever occurs first.

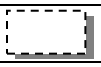
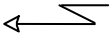
- Manual Calls

The automatic redialling attempts shall be stopped whenever an incoming call is accepted or an outgoing call is manually initiated. The automatically-dialled number shall not be considered unobtainable in this case.

- Blacklist

An unobtainable number shall be entered into a “blacklist” of length 8, and automatic redialling shall not be available for the number. It shall be removed from the “blacklist” only when a successful (manual) call is made to the dialled number, or when the number is manually removed from the list by the user.

Appendix B: Main OPM concepts, their symbols, and their meaning

Concept Name	Symbol	Concept Meaning
Informatical, systemic object		A piece of information
Physical, systemic object		An object which consists of matter and/or energy
Informatical, environmental object		A piece of information which is external to the system
Physical, environmental object		An object which consists of matter and/or energy and is external to the system
Process		A pattern of transformation that objects undergo
Initial/Regular/Final state		An initial/regular/final situation at which an object can exist for a period of time
Characterization		A fundamental structural relation representing that an element exhibits a thing (object/ process)
Aggregation		A fundamental structural relation representing that a thing (object/process) consists of one or more things
General structural link		A bidirectional or unidirectional association between things that holds for a period of time, possibly with a tag denoting the association semantics
Enabling event link		A link denoting an event (such as data change or an external event) which triggers (tries to activate) a process. Even if activated, the process does not change the triggering object.
Consumption event link		A link denoting an event which triggers (tries to activate) a process. If activated, the process consumes the triggering object.
Enabling condition link		A link denoting a condition required for a process execution, which is checked when the process is triggered. If the condition does not hold, the next process (if any) tries to execute.
Consumption conditional link		A link denoting a condition required for a process execution. If activated, the process consumes the conditional object.
Agent link		A link denoting that a human agent (actor) is required for triggering a process execution
Instrument link		A link denoting that a process uses an object without changing it. If the object is not available (possibly in a specific state), the process waits for its availability.
Effect link		A link denoting that a process changes an object
Consumption/Result link		A link denoting that a process consumes/yields an object
Invocation link		A link denoting that a process triggers (invokes) another process when it ends
XOR connection		A connection between procedural links denoting that exactly one of the process incoming/outgoing links is applicable (active) in a single execution of the process