OPCAT – A BIMODAL CASE TOOL FOR OBJECT-PROCESS BASED SYSTEM DEVELOPMENT

Dov Dori, Iris Reinhartz-Berger, Arnon Sturm

Technion, Israel Institute of Technology, Technion City, Haifa 32000, Israel

Emails: {dori@ie, ieiris@tx, sturm@tx}.technion.ac.il

Keywords: CASE tools, software engineering, development process, Object-Process Methodology, simulation.

Abstract: Object-Process CASE Tool (OPCAT), which supports system development using Object-Process Methodology, meets the challenges of next generation CASE tools by providing a complete integrated software and system development environment. The main reasons for which CASE tools have spread at a lower pace than expected are their limited support of a particular method, high cost, lack of measurable returns, and unrealistic user expectations. Although many CASE tools implement familiar methods, their consistency checking and simulation capabilities are limited, if not inexistent, and the syntax and semantics of their graphic notations may not be clear to novice users. Based on two human cognition principles, OPCAT enables balanced modeling of the structural and behavioral aspects of systems in a single view through a bimodal visual-lingual representation. Due to this intuitive dual notation, the resulting model is comprehensible to both domain experts and system architects engaged in the development process. Due to OPM formality, OPCAT also provides a solid basis for implementation generation and an advanced simulation tool, which animates system behavior. This paper presents OPCAT and demonstrates its unique features through a small case study of a travel management information system.

1 INTRODUCTION

Computer-Aided Software Engineering (CASE) tools provide technologies that support automated or semi-automated assistance for software development (Banker and Kauffman, 1991). CASE tools aim at reducing time and cost of software development; increasing the productivity and quality of documentation, analysis, and design; facilitating system maintaining, debugging, and testing; aiding project management; and preserving consistency among various development lifecycle steps (McMurtrey et. al., 2000). These benefits are expected to be achieved by automating the development process, which ideally encompasses the entire system development lifecycle, including requirement managing, consistency checking, implementation generating, and system testing.

As the number of analysis and design methodologies increased, so did the number and variety of their supporting CASE tools. For example, UML (OMG, 2001), which is the industry standard of object-oriented software systems development, has over 96 supporting CASE tools (Object by Design, 2002). The importance and advantages of using CASE tools in the system development process have prompted the IEEE-SA Standards Board to adopt the industrial standard ISO/IEC 14102: 1995, which is concerned with the evaluation and selections of CASE tools, including such issues as the introduction of a new CASE tool into an organization. In spite of this attention and CASE tool vendor development and marketing efforts, CASE tools have spread at a lower pace than expected. The main reason for this is that most available CASE tools support a specific development method, which cannot be easily tailored to the needs of a particular organization and its software developers. In addition, adopting CASE tools within organizations is often expensive, as it includes purchasing or leasing the product, maintaining it, and training the developers to use it. The cost, lack of measurable returns, and lack of fulfillment of unrealistic expectations have discouraged managers from introducing such tools into their companies. Even when decision makers acquire CASE tools, system and software developers within the organization rarely use them.

In this paper, we introduce OPCAT (Object-Process CASE Tool) as an integrated system development software environment. OPCAT supports system development using Object-Process Methodology (OPM) (Dori, 2002), which weaves the object-oriented paradigm with the process-oriented approach. To enhance model legibility and comprehension, OPM uses two semantically equal formalisms: a visual diagramming tool, called Object-Process Diagram (OPD), and a textual counterpart, called Object-Process Language (OPL). Making use of the modality human cognition principle (Mayer, 2001), OPM engages the power of "both sides of the brain" – the visual interpreter and the lingual one.

Being an OPM-based CASE tool, OPCAT enjoys the advantages of supporting most of the system development lifecycle tasks, starting from requirement analysis, through system design and implementation, to system testing, simulation, and validation. Since OPM enables modeling system dynamics and control structures, such as events, conditions, branching, and loops, the generated implementation can definitely be more advanced than a mere standard skeleton code. Moreover, OPM's ability to capture the system's structure and behavior in a single view also enhances OPCAT simulation capabilities and makes it most suitable for interactive testing and validation.

The rest of the paper is organized as follows. In Section 2 we review the literature related to CASE tools and their use. In Section 3, we briefly introduce OPM. Section 4 lists the main features of OPCAT which make it a complete system development and lifecycle support tool. We exemplify these features through a small case study of a travel management system. Finally, in Section 5, we discuss work in progress and future development plan.

2 CASE TOOL UTILIZATION

CASE tools have been developed with the objective of assisting developers in producing high quality software systems and products. To this end, CASE tools are designed to relieve the system architects and developers from mundane software engineering activities, leaving them more time to focus on the non-trivial, insight- and creativity-demanding tasks. Over the years, several studies have surveyed the way organizations use CASE tools. Lending and Chervany (1998) found out that “it was difficult to find companies using CASE tools.” Even in the companies that did use CASE tools, the extent of their deployment was very small. The CASE tool features that these companies employed were divided into two groups: analysis functionality (e.g., testing for consistency between a process model and a data model), and transformation functionality (e.g., generating executable code in several languages). The overall result for using features from both functionalities was low.

McMurtrey et. al. (2000) surveyed the use of CASE technology inquiring professionals from different company types (insurance, manufacturing, consulting firms, etc.). They focused on the most popular features that CASE tools possessed and the gap between them and the developer needs. The features most often cited as being needed and used were the ability to represent a design in terms of data models and process or flow models. This reflects the fact that representing the model's structure and behavior aspects is the most useful aspect of current CASE tools.

In order to overcome some of the CASE tools flexibility drawbacks, a new type of CASE tools, called meta-CASE tools, or Computer Aided Method Engineering (CAME) tools, was introduced. These tools feature flexible metamodeling facilities that users can reconfigure to support whatever metamodel they wish to deploy. Examples of such tools are MetaEdit+ (Talvanen, 2002) and $AToM^3$ (Lara and Vengheluwe, 2002). However, these tools are not widely used, since employing them is not a trivial task.

Method engineering and CASE tool designers have paid little, if any, attention to the human cognition theory. For example, the cognitive theory has shown that the human information processing system involves separate channels for processing visual and verbal material, and that the processing capability of each channel is quite limited (Mayer, 2001). The existing CASE tools address only the visual channel, neglecting the verbal one.

3 OBJECT-PROCESS METHODOLOGY (OPM)

Object-Process Methodology (OPM) (Dori, 2002) is a holistic approach to the study and development of systems. It integrates the object-oriented and process-oriented paradigms into a single frame of reference. Structure and behavior, the two major aspects that each system exhibits, co-exist in the same OPM view without highlighting one at the expense of suppressing the other.

The elements of the OPM ontology are entities (things and states) and links. A *thing* is a generalization of an *object* and a *process*. *Objects* are (physical or informatical) things that exist, while *processes* are things that transform objects. *Links* can be structural or procedural. *Structural links* express static relations between pairs of entities. *Procedural links* connect entities to describe the behavior of a system. The behavior is manifested in three major ways: processes can transform objects; objects can enable processes, and objects can trigger events that invoke processes.

Two semantically equivalent modalities, one graphic and the other textual, jointly express the same OPM model. A set of inter-related Object-Process Diagrams (OPDs) constitute the graphical, visual OPM formalism. Each OPM element is denoted in an OPD by a symbol, and the OPD syntax specifies correct and consistent ways by which entities can be linked. The Object-Process Language (OPL), defined by a grammar, is the textual counterpart modality of the graphical OPD-set. OPL is a dual-purpose language, oriented towards humans as well as machines. Catering to human needs, OPL is designed as a constrained subset of English, which serves domain experts and system architects engaged in analyzing and designing a system. Every OPD construct is expressed by a semantically equivalent OPL sentence or phrase. Designed also for machine interpretation, OPL provides a solid basis for automatically generating the designed application. This dual representation of OPM increases the processing capability of humans.

Another advantage of OPM is its complexity management mechanisms. OPM offers three refinement/abstraction mechanisms: (1) *unfolding/ folding* is used for refining/abstracting the structural hierarchy of a thing; (2) *in-zooming/out-zooming* exposes/hides the inner details of a thing within its frame; and (3) *state expressing/suppressing* exposes/ hides the states of an object. Using flexible combinations of these mechanisms, OPM enables specifying a system to any desired level of detail without losing legibility and comprehension of the resulting specification. The complete OPM system specification is a set of OPDs and their corresponding OPL paragraphs.

OPM has been applied and tested in various domains, including real-time systems (Peleg and Dori, 1999) and Web applications (Reinhartz-Berger et. al. 2002). Developing systems of this nature is a complex task that involves modeling their intertwined structural and behavioral aspects.

4 OPCAT FEATURES: A CASE BASED DEMONSTRATION

OPCAT[1] has been in development as an academic project. It is designed to support the entire system development lifecycle through OPM. The two main benefits of OPCAT over existing object-oriented CASE tools are its bimodal graphic-textual single view representation and its simulation capability. The bimodal representation of OPCAT increases OPM accessibility to heterogeneously skilled users engaged in the system development process. The intuitive, bimodal model representation enables development teams consisting of system architects and domain experts to jointly engage in the development process on an ongoing basis. Such collaboration, which is not feasible with other CASE tools, is highly desirable, because it enables requirements to be put to test while modeling. Improving the system documentation quality is yet another benefit of OPCAT's bimodal representation, since the textual representation provides the system documentation.

OPCAT's simulation capability enables “running” a system model, testing its functionality against the requirement specifications, and debugging them at the model level, prior to the beginning of the implementation phase. In the rest of this section we demonstrate these OPCAT capabilities through a case study of a travel management system. The system manages company employee professional travels, including travel request, approval, finance and expense reporting.

4.1 The Bimodal Graphic-Text Representation

Catering to the modality principle of cognitive theory, OPCAT enables modeling systems graphically via an OPD-set and textually using OPL, a subset of English. OPCAT automatically translates an OPD-set into its equivalent OPL paragraph and vice versa.[2] This way, users who are not familiar with the graphic notation of OPM can validate their specifications by inspecting the OPL sentences, which are automatically generated on the fly in response to the user's graphic input. Another cognitive principle – the limited channel capacity (Mayer, 2001) – is addressed in OPM through the abstraction/refinement mechanisms. These provide for creating diagrams and corresponding OPL paragraphs that are limited in size, thereby avoiding information overload and enabling comfortable human processing. The relatively small set of OPD symbols and corresponding OPL sentence types increases the accessibility of OPM to both system architects and domain experts. The automatic translation into an OPL script also improves the documentation quality of the developed system. The automatic implementation generation, currently under development, will ensure that the specification designed by the system architects and endorsed by the domain experts is indeed reflected without any translational gap in the actual system.

[1] OPCAT can be downloaded free from http://iew3.technion.ac.il/~dori/opcat/index-continue.html

[2] As of writing this paper, the text-to-graphics direction is not yet fully operational.

Figure 1, which is a snapshot of an OPCAT 2 screen, shows the bimodal OPD-OPL representation for the travel management system. The graphic window in the upper part of the screen shows the top-level OPD, while the lower part of the screen is the text window, which contains the equivalent OPL paragraph. Interpreting the OPD or the OPL paragraph, the model specifies that the **Travel Managing** process is handled by the **Employee**, which is an environmental (dashed) and physical (shadowed) object, linked to **Travel Managing** via an agent link. The corresponding OPL sentence that expresses this is: "**Employee,** which is environmental and physical**,** handles **Travel Managing.**" **Travel Managing** is also triggered by the physical and environmental **Clock**, which generates external timing events. **Report** and **Travel Document Set** (which, as the shadow denotes, is physical) are the artifacts resulting as objects from the execution of **Travel Managing**.

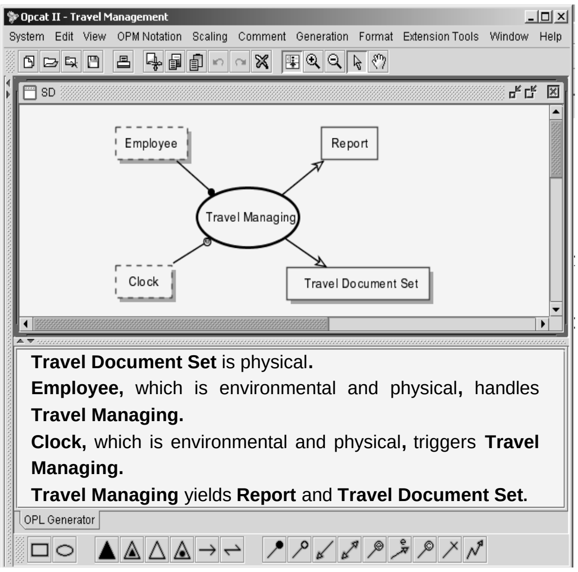

Figure 1.The OPCAT GUI showing the top-level specification of the travel management system

OPCAT 2 performs extensive syntax checking, starting from the validation of simple constraints, such as checking that two objects are not linked via a procedural link, continuing with complex constrains, such as disallowing a loop within a generalization-specialization hierarchy in an OPD, and ending with inter-OPD consistency checking operations.

To specify the details of the behavior of **Travel Managing**, the developer can use the in-zooming refinement mechanism. Applying this refinement on **Travel Managing** in Figure 1 yields a new OPD shown in Figure 2(a), titled "Travel Managing in-zoomed." The in-zoomed **Travel Managing** process appears enlarged in the center of the newly generated diagram. All the entities connected to **Travel Managing** in the top level specification are also connected to it in the new OPD with the same link types.

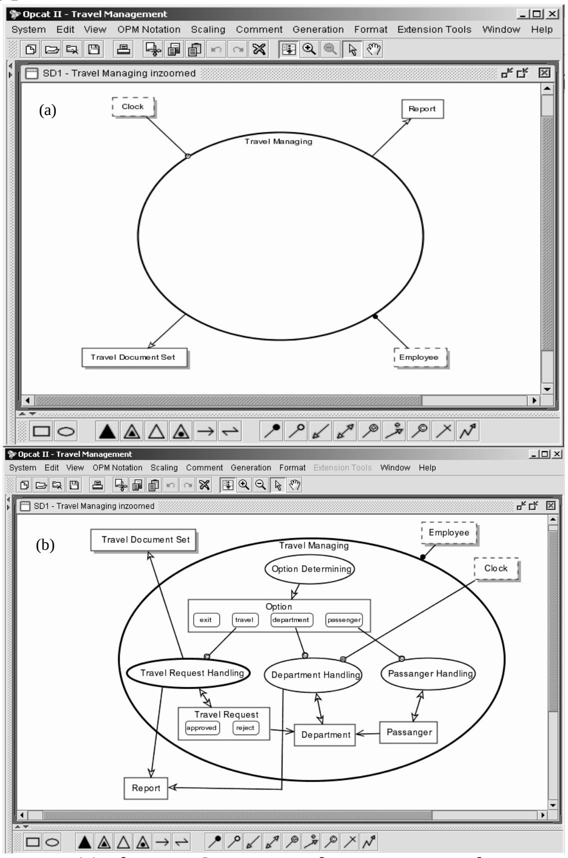

Figure 2. (a) The new OPD, created in response to the user's in-zooming operation on the Travel Managing process. (b) The OPD after the user has filled in details within the in-zoomed Travel Managing process.

The developer can now specify the subprocesses of **Travel Managing** and any pertinent interim objects within its elliptical frame, as shown in Figure 2(b). This refinement specifies that through the **Option Determining** subprocess, the **Employee** first chooses between the **Option** states **travel**, **department**, **passenger**, and **exit**. This selection is a condition to the occurrence of the appropriate process, which can be one of **Travel Requesting**, **Department Handling**, **Passenger Handling**, or nothing.

4.2 Simulation and Dynamic System Testing

Being both object- and process-oriented, OPM enables designing the structural and behavioral aspects of a system in the same view. This fact enables

OPCAT to visually simulate the behavior of the system being developed enabling system architects to dynamically examine the system at any stage of its development and validate with the domain experts that it addresses the client requirements and expectations.

Presenting live animated demonstrations of system behavior reduces the number of design errors percolated to the implementation phase. Both static and dynamic testing help in detecting discrepancies, inconsistencies, and deviations from the intended goal of the system. As part of the dynamic testing, the simulation enables designers to track each of the system scenarios (also known as use cases in UML terminology) before writing a single line of code. Any detected mistake or omission is corrected at the model level, saving costly time and efforts required within the implementation level. Avoiding and eliminating design errors as early as possible in the system development process and keeping the documentation up-to-date contribute to shortening the system's delivery time ("time-to-market").

Although some UML supporting CASE tools provide simulation tools (for example, Rhapsody by I-Logix), the lack of a single clean formalism for expressing processes in UML and the fact that the dynamic views are separate from the static ones make such simulations much less comprehensible, as they can only run on a subset of the nine UML diagram types, overwhelming the limited human cognitive channels and making it extremely difficult to grasp the behavior of the system in its entirety.

OPCAT simulation is performed graphically on the model itself. It is affected by process duration, state duration, and reaction time. After determining these parameters, the designer may manually activate entities, such as agents or instruments, which are connected to processes via external event links. By default, all the objects that are not created by processes in the model are active, but the user can override this. For example, in the initial situation of the travel management system, only **Employee** and **Clock** are active (grayed). These objects had existed before **Travel Managing** started. The process itself is not active, because no event has triggered it yet. The user can then start, stop, pause, continue the simulation, set up breakpoints within the system model, run the simulation forward or backward any specified number of steps, or track it step by step. The simulation algorithm determines the next step according to process activation rules derived from OPM semantics. The guidelines of these rules are that a process becomes active when it is internally or externally activated and its pre-condition set holds. After executing the process, its post-condition set holds.

In order to invoke the **Travel Managing** process, the user has to activate the agent link from **Employee** to **Travel Managing**, simulating the employee action in the real system. Consequently, **Travel Managing** and two of its internal objects, **Department** and **Passenger**, which already exist, become active. **Option** and **Travel Request** do not become active yet, since they are created by the system in this scenario. When the in-zoomed **Travel Managing** becomes active, it activates also its first sub-process, **Option Determining**, for an interval of time determined by the process duration parameter. When **Option Determining** terminates, **Option** becomes active and **Option Determining** reverts to be non-active, as shown in Figure 3. Since **Option** has no initial or default states, the simulation must wait for the user to manually select a state, simulating the **Employee** choice in the real system. This way, designers can selectively simulate use cases within the modeled system. Assuming that the designer activated **travel**, the simulation can continue to the next step, which is to activate the **Travel Requesting** process. The simulation algorithm now examines the pre-conditions of this process: **Option** needs to be in its **travel** state. Since this pre-condition holds, **Travel Requesting** is executed, creating **Travel Request**, **Travel Document Set**, and **Report**. Since this is the last subprocess of the **Travel Managing** super-process, **Travel Managing** terminates, ending the simulation at the situation in which only **Employee**, **Clock**, **Report**, and **Travel Document Set** are active.

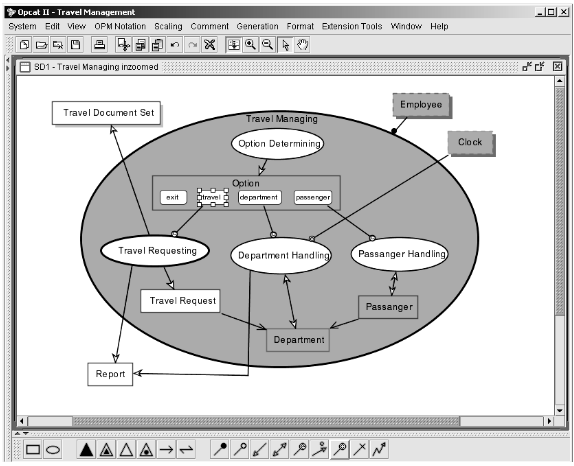

Figure 3. The situation of the travel management system after the **Option Determining** sub-process of **Travel Managing** has terminated and **Option** was generated

5 SUMMARY AND WORK IN PROGRESS

OPCAT has been presented as an integrated system development environment that exhibits a number of unique features. OPCAT implements two important cognitive theory principles: the modality principle and the limited channel capacity principle. To implement the modal principle, OPCAT provides a dual, graphic and textual, model representation. The human limited channel capacity is addressed by implementing the various abstraction/refinement mechanisms that OPM offers.

OPCAT has advanced simulation capabilities. Simulations help visualize the operation of the system at any level of detail, providing a powerful tool for early error detection and correction. Although some UML supporting CASE tools provide simulation tools, the lack of a single formalism for expressing processes in UML and the fact that the dynamic views are separate from the static ones make such simulations less tractable and less comprehensible.

OPCAT has been studied and used in an undergraduate system analysis course for the past two years. Students' responses to OPCAT are enthusiastic. They indicate its reliability, user friendliness, ease of use, and accessibility to untrained users. OPCAT is undergoing major expansion. The following work in progress is currently under way.

- Analysis and design document generation: OPCAT document generator facilitates selective generation of general information, OPDs, OPL paragraphs, and element dictionary. The documents are produced according to user-defined templates.
- Implementation generation: The implementation generator is designed to support conversion rules to various target languages. Using OPL grammar, system developers will have to define once the translation rules from each OPL template to the target languages, and the generic implementation generator will automatically generate the system implementation from its OPL script. Since OPM describes also the behavioral aspects of systems, the generated implementation will be much richer than just a skeleton code.
- OPM-to-UML conversion: Since UML is the standard modeling notation within the software engineering community, we are developing a conversion utility from OPM to UML (uses case, class, sequence, Statecharts, activity, and deployment diagrams) and vice versa. This is done using XML Metadata Interchange (XMI) standard.
- Future features, aimed at further enhancing the usability of OPCAT, include support of automatic layout, a requirement management module, automatic test case generation, configuration management, intelligent knowledge base querying, and a multi-user version, which enables collaboration of project teams.

Acknowledgements: The authors would like to acknowledge the contribution of many past and present students to the development of OPCAT. We are especially grateful to Yevgeny (Zhenya) Yaroker for setting a firm basis for OPCAT and his continuous work on the project. We thank Larisa Shmerling for the development of the simulation tool within OPCAT and to Sergey Krutyolkin for his part in implementing of the OPL generator of OPCAT.

REFERENCES

Banker, R.D. and Kauffman, R.J. 1991. Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study. MIS Quarterly, 15 (3), pp. 375-401.

Dori, D. 2002. Object-Process Methodology - A Holistic Systems Paradigm. Springer Verlag.

Lara, J. and Vangheluwe, H., 2002. Using AToM^3 as a Meta-CASE Tool. Proc. of the 4th Int. Conference On Enterprise Information Systems (ICEIS’2002),.

Lending , D. and Chervany N.L., 1998. The Use of CASE Tools. Proc. of the Conference on Computer Personnel Research, pp. 49-58.

Mayer, R.E., 2001. Multimedia Learning. Cambridge University Press.

McMurtrey, M. E., Teng, J.T.C., Grover, V., and Kher, H. V., 2000. Current utilization of CASE technology: lessons from the field. Industrial Management & Data Systems, 100 (1), pp. 22-30.

Object by Design, 2002. UML Modeling Tools, http://www.objectbydesign.com/tools/umltools_byPlatform.html.

Object Management Group (OMG), 2001. UML 1.4 - UML Semantics.http://cgi.omg.org/docs/formal/01-09-73.pdf.

Peleg, M. and Dori, D., 1999. Extending the Object-Process Methodology to Handle Real-Time Systems. JOOP, 11 (8), pp. 53-58.

Reinhartz-Berger, I. Dov Dori, and Shmuel Katz, 2002, OPM/Web – Object-Process Methodology for Developing Web Applications. ASE, 13, pp. 141–161.

Talvanen, J. P., 2002. Domain-Specific Modelling: Get your Products out 10 Times Faster. Real-Time & Embedded Computing Conference, http://www.metacase.com/papers/Domain-specific_modelling_10X_faster_than_UML.pdf