Abstract
Day-to-day experience suggests that it is not enough to approach a complex design armed with design tips, guidelines, and hints. Developers must also be able to use proven solutions emerging from the best design practices to solve new design challenges. Without these, the designer will not be able to properly apply guidelines or take full advantage of the power of technology, resulting in poor performance, poor scalability, and poor usability. Furthermore, the designer may “reinvent the wheel” when attempting to implement a design solution. A number of design problems continue to arise, such as: (1) decoupling the various aspects of interactive systems (for example, business logic, the UI, navigation, and information architecture; and (2) isolating platform specifics from the concerns common to all interactive systems. In the context of a proposal for a Pattern-Oriented and Model-Driven Architecture (POMA) for interactive systems, this paper identifies an extensive list of patterns categories, and types of models aimed at providing a pool of proven solutions to these problems. The models of patterns span several levels of abstraction, such as domain, task, dialog, presentation, and layout. The proposed POMA architecture illustrates how several individual models can be combined at different levels of abstraction into heterogeneous structures, which can then be used as building blocks in the development of interactive systems.

First, we describe the architectural levels and the categories of patterns, as well as the various relationships between patterns; second, we propose five categories of models to address the problems described above which are associated with creating an interactive system. Third, we present the proposed POMA architecture. Fourth, we present a case study to illustrate and clarify the core ideas of our approach and its practical relevance.

Keywords: Patterns, Models, Architecture, Interactive Systems, Composition, Mapping, Transformation, POD, MDA, POMA

1. Introduction

During the past two decades, research on interactive system and user interface (UI) engineering has resulted in a set of design principles and development frameworks which constitute a major contribution, not only to facilitate the development and maintenance of interactive systems, but also to promote the standardization, portability, and ergonomic “usability” (ease of use) of the interactive systems being developed. Some of these principles are:
A precise definition of the UI aimed at: (i) presenting the output to the user; (ii) gathering user entries to transmit them to the interactive system procedures that will treat them; (iii) handling the dialog sequence;

- The separation of concerns, especially decoupling the UI from the system semantics;
- The definition of reusable and standardized UI components;
- Decentralization of dialog management, help, and errors across the various components of an interactive system;
- Programming driven by events.

Indeed, an interactive system is a program with which the user engages in conversation (dialog) in order to accomplish tasks. An interactive system consists of two parts: the software, which is referred to as the interactive application, and the hardware, which supports the execution of the software part.

The software (interactive application) can, in turn, be divided into two sub-parts: the UI, and the algorithms that constitute the semantics of the interactive system.

The hardware in an interactive system consists of input and output devices and various managers (device drivers) which provide physical support to the execution of the interactive application.

At the same time, a UI can be seen as a means by which the user and the machine can exchange data. For example, the screen on which data are displayed is a medium for user-machine interaction and for feedback in response to the user’s actions. Therefore, a UI is part of an interactive application which:

- Presents the output to the user,
- Collects the user’s inputs and transmits them to the interactive system that treats them, and
- Handles the dialog sequence.

Based on these principles, several interactive system architectural models have been introduced. Buschmann et al. define architectural models as [41]: “the structure of the subsystems and components of a system and the relationships between them typically represented in different views to show the relevant functional and non functional properties.” This definition introduces the main architectural components (for instance, subsystems, components, and connectors) and covers the ways in which to represent them, including both functional and nonfunctional requirements, by means of a set of views.

A number of architectures specific to interactive systems have been proposed, e.g. Seeheim model [55; 48], Model-View-Controller (MVC) [43], Agent Multi-Faceted (AMF) [54], which is an extension of MVC, Arch/Slinky [47], Presentation Abstraction Control (PAC) [44; 45], PAC-Amadeus, and Model-View-Presenter (MVP) [27]. Most of these architectures consist of three main elements: (1) abstraction or model; (2) control or dialog; and (3) presentation. Their goal is to improve and facilitate the design of interactive systems. However, even though the principle of separating an interactive system into components has its design merits, it can also be a source of serious adaptability and usability problems in systems which provide fast, frequent, and intensive semantic feedback: the communication between the view and the model makes the interactive system highly coupled and complex.

Among the problems presented by the architectures quoted above are:
• No guideline is provided to encourage the designer to consider other aspects of the
dialog that are important to the user, such as assistance or error-handling;
• Lack of facilitation of the use of constraints for the design and description of the
interface, when these constraints are a great importance to the designer [49; 50; 51;
52; 53; 46];
• The architectural models are poorly located in relation to the life cycle of the UI,
which can lead, in particular, to difficulties concerning the passage of the problem
analysis (analysis of user needs), expressed generally in terms of tasks and
interaction sequences, and to the concepts put forward by these architectures
(agents, presentation components, dialog components).

Architectural patterns have been proposed to alleviate some of these problems, and indeed
were introduced based on the following observation by Alexander: “Each pattern describes a
problem that occurs constantly in our environment, and thus describes the heart of the solution
to this problem, in such a way that we can use this solution millions of times, but never do it
twice the same way” [40]. Such a pattern provides, on one level, a pool of proven solutions to
many of the recurring problems listed above. The Pattern-Oriented Software Architecture is an
example of a new approach which shows how to combine individual patterns into
heterogeneous structures and, as such, can be used to facilitate a constructive instantiation of
a system architecture.

Patterns provide various benefits, such as:
• Well-established solutions to architectural problems,
• Help in documenting architectural design decisions, and
• Facilitation of communication between users through a common vocabulary.

However, we note that the emergence of patterns in the architectural development of
interactive systems has not solved some problems associated with this development.

Among the challenging problems we address in this paper are the following:
(a) Decoupling of the various aspects of interactive systems, such as business logic, UI,
navigation, and information architecture;
(b) Isolation of the platform-specific problems from the concerns common to all
interactive systems.

In 2001, the Object Management Group introduced the Model-Driven Architecture (MDA)
initiative as an architecture to system specification and interoperability based on the use of
formal models (i.e. defined and formalized models). The main idea behind MDA is to specify
business logic in the form of abstract models. These models are then mapped (partly
automatically) according to a set of transformation rules to different platforms. The models are
usually described by UML in a formalized manner which can be used as input for tools to
perform the transformation process.

Indeed, a model is a formal description of some key aspects of a system, from a specific
viewpoint. As such, a model always presents an abstraction of the "real" thing, by ignoring or
deliberately suppressing those aspects that would not of interest to a user of that model.
Different modeling constructs focus attention by ignoring certain things [35]. For example, an
architectural model of a complex system might focus on its concurrency aspects, while a
financial model of a business might focus on projected revenues. Model syntax includes graphical or tabular notations and text.

D’Souza [35] has identified key opportunities and modeling challenges, and illustrated how the “model” and the “architecture” of MDA could be used to enable large-scale model-driven integration. The advantages of the models are as follows:

- They make it easier to validate the correctness of a model.
- They make it easier to produce implementations on multiple platforms.
- Integration/interoperability across platforms is better defined.
- Generic mappings/patterns can be shared by many designs.
- They constitute an interactive system of tool-supported solutions.

However, we note that the model-driven approach has some weaknesses as well:

- MDA does not provide a standard for the specification of mappings: different implementations of mappings can generate very different code and models, which can create dependencies between the system and the mapping solution used.
- Designers must take into account a diversity of platforms which exhibit drastically different capabilities. For example, Personal Digital Assistants (PDAs) use a pen-based input mechanism and have average screen size in the range of 3 inches.
- The architectural models must be located and compared to the life cycle of the UI: in particular, difficulties may arise related to the problem analysis (analyzing user needs), expressed generally in terms of tasks and interaction sequences, and to the concepts proposed by these architectures (agents, presentation components, and dialog components).

Our research goal can be stated as follows: “a new architecture to facilitate the development and migration of interactive systems while improving their usability and quality.” To pursue this goal, it is necessary to define a systematic architecture, supported by a CASE tool, to glue patterns together. In this paper, we identify some of the fundamentals of such an architecture and we present an architecture called the Pattern-Oriented and Model-Driven Architecture (POMA) (Figure 14). We also present an evaluation of the feasibility of some phases of this architecture, such as composition and mapping patterns, and transformation models to create platform-independent models (PIM) and platform-specific models (PSM). Figure 1 summarizes the architectural patterns and models that were combined to obtain the POMA architecture.
In this paper, we propose an architectural model which combines two key approaches: model-driven and pattern-oriented. First, we describe architectural levels and categories of patterns, as well as the various relationships between patterns. These relationships are used next to combine and map several categories of patterns to create a pattern-oriented design for an interactive system, as well as to show how to generate specific implementations suitable for different platforms from the same pattern-oriented design. Second, we propose five categories of models (Domain model, Task model, Dialog model, Presentation model, and Layout model) which address problems such as: (a) decoupling the various aspects of interactive systems, such as business logic, UI, navigation, and information architecture; and (b) isolating the platform-specific problems from the concerns common to all interactive systems. Third, the proposed Pattern-Oriented and Model-Driven Architecture (POMA) illustrates how the individual models mentioned above can be combined at different levels of abstraction into heterogeneous structures to be used as building blocks in the development of interactive systems. Fourth, we present a case study to illustrate our approach and its practical relevance. Finally, we provide a conclusion on the research carried out and comment on its evolution in the future.

2. Patterns and Pattern-Oriented Architecture

2.1. About Patterns

The idea of using patterns in system design and engineering is not new. It has its roots in the popular work, Gang of Four [2]. Collections of patterns include patterns for UI (UI) design [3; 4; 5], patterns for navigation in a large information architecture, and patterns for visualizing and presenting information. More recently, the concept of the usability pattern has been introduced and discussed as a tool for ensuring the usability of developed systems [16; 17; 18]. A usability pattern is a proven solution for a User Centered Design (UCD) problem that recurs in different contexts. The primary goal of usability patterns in general is to create an inventory of solutions to help UI designers to tackle UI development problems which are common and difficult to solve [19].

2.2. Architectural Levels and Categories of Patterns

A number of pattern languages have been suggested, such as in Van Duyne’s “The Design of Sites” [10], Welie’s “Interaction Design Patterns” [5], and Tidwell’s “UI Patterns and Techniques” [3]. In addition, specific languages have been proposed, such as in Laakso’s “UI Design Patterns” [31] and the UPADE Language [29]. Moreover, various specialized collections of patterns have been published, including patterns for Web page layout design [3; 4; 5], patterns for navigation around information architectures, and patterns for visualizing and presenting information.

In our work here, we illustrate how these existing collections of patterns can be used as building blocks in the context of the proposed six-layer architecture.

An informal survey conducted in 2004 by the HSCE Research Group at Concordia University identified at least six architectural levels and six categories of patterns which can be used to create a pattern-oriented interactive system architecture. Table 1 illustrates these six levels of architecture for an interactive system, including the corresponding categories of patterns, and gives examples of patterns in each category.
<table>
<thead>
<tr>
<th>Architectural Level and Category of Patterns</th>
<th>Examples of patterns</th>
</tr>
</thead>
</table>
| **Information** This category of patterns describes different conceptual models and architectures for organizing the underlying content across multiple pages, servers, and computers. Such patterns provide solutions to questions such as which information can or should be presented on which device. This category of patterns is described in [39]. | - Reference Model pattern  
- Data Column pattern  
- Cascaded Table pattern  
- Relational Graph pattern  
- Proxy Tuple pattern  
- Expression pattern  
- Schadler pattern  
- Operator pattern  
- Renderer pattern  
- Production Rule pattern  
- Camera pattern |
| **Interoperability** This category of patterns describes decoupling the layers of an Interactive system; in particular, between the content, the dialog, and the views or presentation layers. These patterns are generally extensions of the Gamma design patterns, such as MVC (Model, View, and Controller) observer and command action patterns. Communication and interoperability patterns are useful for facilitating the mapping of a design between platforms. | - Adapter pattern  
- Bridge pattern  
- Builder pattern  
- Decorator pattern  
- Façade pattern  
- Factory pattern  
- Method pattern  
- Mediator pattern  
- Memento pattern  
- Prototype pattern  
- Proxy pattern  
- Singleton pattern  
- State pattern  
- Strategy pattern  
- Visitor pattern |
| **Visualization** This category of patterns describes different visual representations and metaphors for grouping and displaying information in cognitively accessible chunks. They mainly define the format and content of the visualization, i.e. the graphical scene, and, as such, relate primarily to data and mapping transforms. | - Favorite Collection pattern  
- Bookmark pattern  
- Frequently Visited Page pattern  
- Navigation Space Map pattern |
| **Navigation** This category of patterns describes proven techniques for navigating within and/or between a set of pages and chunks of information. This list is far from exhaustive, but helps to communicate the flavor and abstraction level of design patterns for navigation. | - Shortcut pattern  
- Breadcrumb pattern  
- Contextual (temporary) horizontal menu at top pattern  
- Contextual (temporary) vertical menu at right pattern  
- Information portal pattern  
- Permanent horizontal menu at top pattern  
- Permanent vertical menu at left pattern  
- Progressive filtering pattern  
- Shallow menus pattern  
- Simple universal pattern  
- Split navigation pattern  
- Sub-sites pattern  
- User-driven pattern  
- Alphabetical index pattern  
- Key-word search pattern  
- Intelligent agents pattern  
- Container navigation pattern  
- Deeply embedded menus pattern  
- Hybrid approach pattern  
- Refreshed shallow vertical menus pattern |
| **Interaction** This category of patterns describes the interaction mechanisms that can be used to achieve tasks and the visual effects they have on the scene; as such, they relate primarily to graphical and rendering transforms. | - Search pattern  
- Executive Summary pattern  
- Action Button pattern  
- Guided Tour pattern  
- Paging pattern  
- Pull-down Button pattern  
- Slideshow pattern  
- Stepping pattern  
- Wizard pattern |
| **Presentation** This category of patterns describes solutions for how the contents or the related services are visually organized into working surfaces, the effective layout of multiple information spaces, and the relationship between them. These patterns define the physical and logical layout suitable for specific interactive systems. | - Carrousel pattern  
- Table Fiber pattern  
- Detail On Demand pattern  
- Collector pattern  
- In place Replacement pattern  
- List Builder pattern  
- List Entry View pattern  
- Overview by Detail pattern  
- Part Selector pattern  
- Tabs pattern  
- Table Sorter pattern  
- Thumbnail pattern  
- View |

Table 1: Architectural levels and categories of patterns
Each of these six categories of patterns is discussed below, and examples are provided.

**Information patterns**

An information pattern, also called an information architectural pattern (Figure 2), expresses a fundamental structural organization or schema of information. It provides a set of predefined subsystems (information spaces or chunks), specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

An information pattern is everything that happens in a single information space or chunk. With another pattern, the content of a system is organized in a sequence in which all the information spaces or chunks are arranged as peers, and every space or chunk is accessible from all the others. This is very common on simple sites, where there are only a few standard topics, such as: Home, About Us, Contact Us, and Products. Information which naturally flows as a narrative, time line, or in a logical order is ideal for sequential treatment. An index structure is like the flat structure, with an additional list of contents. An index is often organized in such a way as to make its content easier to find. For example, a list of files in a Web directory (the index page), an index of people's names ordered by last name, etc. Dictionaries and phone books are both very large indices.

The Hub-and-Spoke pattern is useful for multiple, distinct, linear workflows. A good example would be an email system, where the user returns to his inbox at several points, e.g. after reading a message, after sending a message, or after adding a new contact. A multi-dimensional hierarchy is one in which there are many ways of browsing the same content. In a way, several hierarchies may coexist, overlaid on the same content. The structure of the content can appear to be different, depending on the user’s task (search, browse). A typical example would be a site like Amazon, which lets you browse books by genre or by title, and also lets you search by keyword. Each of these hierarchies corresponds to a property of the content, and each can be useful, depending on the user’s situation. A strict hierarchy is a specialization of the multi-dimensional hierarchy, and describes a system where a lower-level page can only be accessed via its parent.
Interoperability patterns

Interoperability patterns are useful for decoupling the way these different categories of patterns are organized, the way information is presented to the user, and the user who interacts with the information content. Patterns in this category generally describe the capability of different programs to exchange data via a common set of exchange formats, to read and write under the same file formats, and to use the same protocols.

Gamma et al. [2] offer a large catalog of patterns for dealing with such problems. Examples of patterns applicable to interactive systems include: Adapter, Bridge, Builder, Decorator, Factory Method, Mediator, Memento, Prototype, Proxy, Singleton, State, Strategy, and Visitor [2].

The Adapter pattern is very common, not only to remote client/server programming, but also to any situation in which there is one class and it is desirable to reuse that class, but where the system interface does not match the class interface. Figure 3 illustrates how an adapter works. In this figure, the Client wants to invoke the method request() in the Target interface. Since the Adaptee class has no request() method, it is the job of Adapter to convert the request to an available matching method. Here, the Adapter converts the method request() call into the Adaptee method specificRequest() call. The Adapter performs this conversion for each method that needs adapting. This is also known as Wrapping.

![Figure 3: Adapter pattern](image)

Visualization patterns

Information visualization patterns allow users to browse information spaces and focus quickly on items of interest. Visualization patterns can help in avoiding information overload, a fundamental issue to tackle, especially for large databases, Web sites, and portals, as they can access millions of documents. The designer must consider how best to map the contents into a visual representation which conveys information to the user while facilitating exploration of the content. In addition, the designer must undertake dynamic actions to limit the amount of information the user receives, while at the same time keeping the user informed about the content as a whole. Several information visualization patterns generally combine in such a way that the underlying content can be organized into distinct conceptual spaces or working surfaces which are semantically linked to one another.

For example, depending on the purpose of the site, users can access several kinds of "pages", such as articles, URLs, products, etc. They typically collect several of these items for
a specific task, such as comparing, buying, going to a page, sending a page to others, etc. Users must be able to visualize their "collection".

The following are some of the information visualization patterns for displaying such collections: Favorite, Bookmark, Frequently Visited Page, Preferences, and Navigable Spaces Map. This category of patterns provides a map to a large amount of content which can be too much to be presented reasonably in a single view. The content can be organized into distinct conceptual spaces or working surfaces which are semantically linked, so that it is natural and meaningful to go from one to another. The map in Figure 4 is an example of this category of patterns.

![Figure 4: The Navigation Spaces Map pattern implemented using Tree Hyperbolic, a sophisticated visualization technique.](image)

**Navigation patterns**

Navigation patterns help the user move easily and in a straightforward manner between information chunks and their representations. They can obviously reduce the user's memory load [7; 1]. See [3; 5; 29; 9] for an exhaustive list of navigation patterns.

The Linear Navigation pattern is suitable when a user wants a simple way to navigate from one page to the next in a linear fashion, i.e. move through a sequence of pages. The Index Browsing pattern is similar, and allows a user to navigate directly from one item to the next and back. The ordering can be based on a ranking. For every item presented to the user, a navigation widget allows the user to choose the next or previous item in the list. The ordering criterion should be visible (and be user-configurable). To support orientation, the current item number and total number of items should be clearly visible. A breadcrumb (Figure 5) is a widely used pattern which helps users to know where they are in a hierarchical structure and to navigate back up to higher levels in the hierarchy. It shows the hierarchical path from the top level to the current page and makes each step clickable.

![Figure 5: Breadcrumb pattern](image)
Interaction patterns

This category of interaction patterns provides basic information on interaction style, mainly on how to use controls such as buttons, lists of items, menus, dialog boxes, etc. This category of patterns is used whenever users need to take an important action that is relevant in the current context of the page they are viewing. Users must be made aware of the importance of the action in relation to other actions on the page or site.

To view/act on a linear-ordered set of items, the Stepping pattern (Figure 6) allows users to go to the next and previous task or object by clicking on the 'next' or 'previous' links. The 'next' link takes the users to the next item in the sequence, while the 'previous' link takes them a step back. It is recommended that a 'next' or 'previous' link be placed close to the object to which it belongs, preferably above the object so that users do not have to scroll to it. Make sure the next/previous links are always placed in the same location, so that users clicking through a list do not have to move the mouse pointer. The convention, at least in western cultures, is to place the 'previous' link on the left and the 'next' link on the right.

Presentation patterns

The authors of technical documents discovered long before interactive systems were invented that users appreciate short "chunks" of information [6]. Patterns in this category, called Presentation patterns, also suggest different ways for displaying chunks of information and ways for grouping them in pages. Presentation patterns also define the look and feel of interactive systems, while at the same time defining the physical and logical layout suitable for specific systems, such as home pages, lists, and tables. For example, how long does it take to determine whether or not a document contains relevant information? This question is a critical design issue, in particular for resource-constrained (small) devices.
Patterns in this category use a grid, which is a technique taken from print design, but which is easily applicable to interactive system design as well. In its strictest form, a grid is literally a grid of X by Y pixels. The elements on the page are then placed on the cell borderlines and aligned overall on horizontal and vertical lines. A grid is a consistent system in which to place objects. In the literature on print design, there are many variations of grids, most of them based on modular and column grids. Often, a mix of both types of grids will be used. An example of a grid in Figure 7 is used to create several dialog box patterns.

![Figure 7: An example of a grid](image)

An example of these types of patterns is the Executive Summary pattern. Our Executive Summary pattern gives users a preview of the underlying information before they spend time downloading, browsing, and reading large amounts of information (Figure 8).

![Figure 8: Example of a structural pattern: Executive Summary](image)

2.3. Pattern Composition

A platform-independent pattern-oriented design exploits several relationships between patterns. Gamma et al. [2] emphasize that defining the list of related patterns as part of the description of a pattern is a key notion in the composition of patterns and their uses. Zimmer [15] implements this idea by dividing the relations between the patterns of the Gamma catalog...
into three types: “X is similar to Y”, “X uses Y”, and “Variants of X use Y”. These types are, in practice, relationships between patterns in a specific context; in other words, they are relationships between instances of patterns. Based on Zimmer’s definitions [15], we define five types of relationships between patterns.

**Similar**

Two patterns (X, Y) are similar, or equivalent, if and only if X and Y can be replaced by each other in a certain composition. This means that X and Y are patterns of the same category and they provide different solutions to the same problem in the same context. As illustrated in Figure 9, the *Index Browsing* and *Menu Bar* patterns are similar. They both provide navigational support in the context of a medium-sized interactive system.

![Figure 9: Similar patterns extracted from the OMG site](image)

**Competitor**

Two patterns (X, Y) are competitors if X and Y cannot be used at the same time for designing the same artifact relationship that applies to two patterns of the same pattern category. Two patterns are competitors if and only if they are similar and interchangeable. For example, the Web patterns *Convenient Toolbar* and *Index Browsing* are competitors (Figure 10). The *Index Browsing* pattern can be used as a shortcut toolbar that allows a user to directly access a set of common services from any interactive system. The *Convenient Toolbar*, which provides the same solution, is generally considered more appropriate.

![Figure 10: Two Competitor patterns](image)
Super-ordinate
A pattern X is a super-ordinate of pattern Y, which means that pattern Y is used as a building block to create pattern X. An example is the Home Page pattern, which is generally composed of several other patterns (Figure 11).

Figure 11: Home Page pattern with sub-ordinate patterns

Sub-ordinate
(X, Y) are sub-ordinate if and only if X is embeddable in Y. Y is also called a super-ordinate of X. This relationship is important in the process of mapping pattern-oriented design from one platform to another. For example, the Convenient Toolbar pattern (Figure 11) is a sub-ordinate of the Home Page pattern for either a PDA or desktop system. Implementations of this pattern are different for different devices.

Neighboring
Two patterns (X, Y) are neighboring if X and Y belong to the same pattern category. For example, the Sequential and Hierarchical patterns are neighboring because they belong to the same category of patterns, and neighboring patterns may include the set of patterns for designing a specific page such as a home page (Figure 11).

2.4. Pattern Mapping
Another component in our architectural framework is the concept of pattern mapping. Pattern mapping is the process of creating a design of specific models for each platform called platform-specific model (PSM) – from the PIM and mapping rules which are described below. Using a desktop system as a starting point, it is possible to redesign it for other platforms. The original set of patterns used in the system is mapped or replaced, in order to redesign and re-implement the system and, in particular, the UI for mobile or Personal Digital Assistant (PDA) systems. Since patterns hold information about design solutions and context of use, platform capabilities and constraints are implicitly addressed in the transformed patterns.
To illustrate pattern mapping, we describe here the effect of screen size on the selection and use of patterns. Different platforms use different screen sizes, and these different screen sizes afford different types and variants of patterns. The problem to resolve when mapping a pattern-oriented design (POD) is how the change in screen size between two platforms affects redesign at the pattern level. The amount of information that can be displayed on a given platform screen is determined by a combination of area and number of pixels. The total difference in information capacity between platforms will be somewhere between these two measures: 20 times the area and 10 times the pixels.

To map the desktop display architecture to the PDA display architecture, the options are as follows:

1. To reduce the size of the architecture, it is necessary to reduce significantly both the number of pages and the quantity of information per page.
2. To hold the architecture size constant (i.e. topics or pages), it is necessary to significantly reduce the quantity of information per page (by a factor of about 10 to 20).
3. To retain all the information in the desktop architecture, it is necessary to significantly increase the size of the architecture, since the PDA can hold less information per page.

The choice of mapping will depend on the size of the architecture and the value of the information:
- For small desktop architectures, the design strategy can be weighted either toward reducing information if the information is not important, or toward increasing the number of pages if the information is important.
- For medium and large desktop architectures, it is necessary to weight the design strategy heavily toward reducing the quantity of information, otherwise the architecture size and number of levels would rapidly explode out of control.

Finally, we can consider mapping patterns and graphical objects in the context of the amount of change that must be applied to the desktop design or architecture to fit it into a PDA format. The following is the list of mapping rules we suggest:

1. **Identical**: No change to the original design. For example, drop-down menus can usually be copied from a desktop to a PDA without any design changes.
2. **Scalable**: Changes to the size of the original design or to the number of items in the original design. For example, a long horizontal menu can be adapted to a PDA by reducing the number of menu elements.
3. **Multiple**: Repeating the original design, either simultaneously or sequentially. For example, a single long menu can be transformed into a series of shorter menus.
4. **Fundamental**: Change the nature of the original design. For example, permanent left-hand vertical menus are useful on desktop displays, but are not practical on most PDAs. In mapping to a PDA, left-hand menus normally need to be replaced with an alternative such as a drop-down menu.

These mapping rules can be used by designers in the selection of patterns, especially when different patterns apply for one platform but not for another, when the cost of adapting or purchasing a pattern is high, or when the applicability of a pattern (knowing how and when to apply a pattern) is questionable.
This list of four mapping rules is especially relevant to the automation of cross-platform design mapping, since the designs that are easiest to map are those that require the least mapping. The category of patterns therefore identifies where human intervention will be needed for design decisions in the mapping process. In addition, when building a desktop design for which a PDA version is also planned, the category of patterns indicates which patterns to use in the desktop design to allow easy mapping to the PDA design.

Figure 12 illustrates some of the navigation design patterns used in the home page of a desktop-based system. Once these patterns are identified in the desktop-based system, they can be mapped or replaced by others in a PDA version.

Figure 12: Patterns extracted from the CBC News site

Figure 13 demonstrates the redesigned interface of the CBC site for migrating to a PDA platform. The permanent horizontal menu pattern at the top (P5) in the original desktop UI were repositioned to a shorter horizontal menu pattern (P5s). In order to accommodate this change on the small PDA screen, the three different horizontal menus had to be shortened, and only important navigation items were used. The keyword search pattern (P13) remains as a keyword search. The permanent vertical menu on the left (P6) was redesigned to a drop-down menu (P15). The drop-down menu in the PDA design also includes the menu headings, “What’s on today?” and “Online features” from the temporary vertical menu (P3) in the original desktop design. Finally, the information portal (P4), which is the first item that captures the user’s attention, was redesigned as a smaller information portal (P4s).

Figure 13: Migration of the CBC site to a PDA Platform using Pattern Mapping
3. Models and Model-Driven Design: Categorization and Transformation

3.1. About Models

As the complexity of interactive systems grows, the role of models is becoming essential for dealing with the numerous aspects involved in their development and maintenance processes. Models allow the relevant aspects of a system to be captured from a given perspective and at a specific level of abstraction. In a model-driven UI design approach, various models are used to describe the relevant aspects of the UI. Many facets exist, as well as related models. Design is an assembly of parts which realizes a specification. A model of a system is a specification of that system and its environment for some purpose. Models consist of a set of elements with a graphical and/or textual representation [56]. The idea behind model-driven design is to create different models of a system at different levels of abstraction, and to use transformations of the models to produce the implementation of the interactive systems.

A number of distinct models have been suggested in the following work:

- the OMG’s Model-Driven Architecture [3, 4, 5, 6, 7],
- Si Alhir’s Understanding the Model Driven Architecture (MDA), Methods & Tools’ [8],
- Paternò’s Model-Based Design and Evaluation of Interactive Systems [22],
- Vanderdonckt’s Task Modeling in Multiple Contexts of Use [23],
- Msheik’s Compositional Structured Component Model: Handling Selective Functional Composition [30],
- Puerta’s Modeling Tasks with Mechanisms [24].

In our work here, we investigate how these existing collections of models can be used as building blocks in the context of the five levels of the proposed POMA architecture.

Our approach focuses on a subset of the proposed models and consists of:

- a domain model,
- a task model,
- a dialog model,
- a presentation model, and
- a layout model.

In the work reported in the next section, we describe how these models can be used at five levels of the proposed POMA architecture to create a model-driven architecture for interactive systems.

3.2. Model Categorization

A categorization of models is proposed next. Examples of models are also presented to illustrate the need to map and/or to transform several types of models to provide solutions to complex problems at the five architectural levels.

Domain Model

The Domain model is sometimes called a business model. It encapsulates the important entities of a system domain along with their attributes, methods, and relationships [13]. Within the scope of UI development, it defines the objects and functionalities accessed by the user.
via the interface. Such a model is generally developed using the information collected during the business and functional requirements stage. The information defines the list of data and features or operations to be performed in various ways, i.e. by different users on different platforms.

The first model-based approaches used a Domain model to drive the UI at runtime. In this context, the Domain model would describe the system in general, and include some specific information for the UI. For example, the Domain model [13] would include:

- a class hierarchy of objects which exist in the system,
- the properties of the objects,
- actions which can be performed on the objects,
- units of information (parameters) required by the actions, and
- pre- and post-conditions for the actions.

Domain models should represent the important entities along with their attributes, methods, and relationships.

Consequently, the only real way to integrate UI and system development is the simultaneous use of the data model. That is why recent model-based approaches include a Domain model known from the system engineering methods. Four other models: Task, Dialog, Presentation, and Layout, have the Domain model as an input.

Figure 14 is an example of implementation of the Login pattern.

![Figure 14: Login view of the system on the laptop platform](image)

Figure 15 is an example of the implementation of the Login pattern.

![Figure 15: Login view of the system on the PDA platform](image)
Task model

The Task model makes it possible to describe how tasks can be performed to reach the user’s goals when interacting with an interactive system [22]. Using this model, designers can develop integrated descriptions of the system from a functional and interactive point of view. Task models are typically tasks and subtasks hierarchically decomposed into atomic actions [23]. In addition, the relationships between tasks are described with the execution order or dependencies between peer tasks. The tasks may contain attributes about their importance, their duration of execution, and their frequency of use.

For our purposes, we reuse the following definition:

A **task** is a goal, along with the ordered set of tasks and actions that would satisfy it in the appropriate context. [13]

This definition highlights the intertwining nature of tasks and goals. Actions are required to satisfy goals. Furthermore, the definition allows the decomposition of tasks into sub-tasks, and there exists some ordering among the sub-tasks and actions. In order to support this definition, we need to add the definitions of goal, action, and artifact:

A **goal** is an intention to change or maintain the state of an artifact (based on [13]).
An **action** is any act which has the effect of changing or maintaining the state of an artifact (based on [13]).
An **artifact** is an object which is essential for a task. Without this object, the task cannot be performed; the state of this artifact is usually changed in the course of performance of a task. Artifacts are real things which exist in the context of task performance. In business, artifacts are modeled as objects and represented in the business model. This implies a close relationship between the Task model and the business model.

With these definitions, we can derive the information that needs to be represented in a Task model. According to [13], the description of one task includes:

- One goal,
- A non-empty set of actions or other tasks which are necessary to achieve the goal,
- A plan of how to select actions or tasks, and
- A model of an artifact, which is influenced by the task.

Consequently, the development of the Task model and the Domain model is interrelated. One of the goals of model-based approaches is to support user-centered interface design. Therefore, they must enable the UI designer to create the various Task models. Three other models: Dialog, Presentation, and Layout, have the Domain and Task models as inputs.

Figure 16 represents the structure of the Task model of the environmental management system. As shown in Figure 16, the Login, Multi-Value Input Form and Find patterns can be used in order to complete the Task model at the lower levels.
Figure 16: Coarse-grained Task model of the environmental management system
**Dialog Model**

This model makes it possible to provide dialog styles to perform tasks and to provide proven techniques for the dialog. The Dialog model defines the navigational structure of the UI. It is a more specific model and can be derived in good measure from the more abstract Task, User, and Domain models.

A dialog model is used to describe the human-computer conversation. It specifies when the end-user can invoke commands, functions, and interaction media, when the end-user can select or specify inputs, and when the computer can query the end-user and present information [26]. In other words, the Dialog model describes the sequencing of input tokens, output tokens, and the way in which they are interleaved. It describes the syntactical structure of human-computer interaction. The input and output tokens are lexical elements. Therefore, in particular, this model specifies the user commands, interaction techniques, interface responses, and command sequences permitted by the interface during user sessions. Two other models, Presentation and Layout, have the Domain, Task, and Dialog models as inputs.

Figure 17 depicts the various dialog view interactions of the environmental management system’s suggested dialog graph structure for the laptop and PDA platforms.

![Figure 17: Dialog Graph of the environmental management system for the laptop and PDA platforms](image-url)
Presentation Model

The Presentation Model describes the visual appearance of the UI [13]. This model exists at two levels of abstraction: the abstract and the concrete. In fact, they define the appearance and the form of presentation of the system on the interactive system providing solutions as to how the contents or related services can be visually organized into working surfaces, the effective layout of multiple information spaces, and the relationship between them. Moreover, they define the physical and logical layout suitable for specific interactive systems such as home pages, lists, and tables.

A Presentation model describes the constructs that can appear on an end-user’s display, their layout characteristics, and the visual dependencies among them. The displays of most systems consist of a static part and a dynamic part. The static part includes the presentation of the standard widgets like buttons, menus, and list boxes. Typically, the static part remains fixed during the runtime of the interactive system, except for state changes like enable/disable, visible/invisible. The dynamic part displays system-dependent data, which typically change during runtime (e.g. the system generates output information, while the end-user constructs system-specific data).

The former provides an abstract view of a generic interface, which represents corresponding Task and Dialog models. Another model, Layout, has the Domain, Task, Dialog, and Presentation models as inputs. Figure 23 portrays the presentation model.

Layout Model

A Layout model constitutes a concrete instance of an interface. It consists of a series of UI components which defines the visual layout of a UI and the detailed dialogs for a specific platform and context of use. There may be many concrete instances of a Layout model which can be derived from Presentation and Dialog models.

This model makes it possible to provide conceptual models and architectures for organizing the underlying content across multiple pages, servers, databases, and computers. It is concerned with the look and feel of interactive systems and with the construction of a general drawing area (e.g. a canvas widget), and all the outputs inside a canvas must be programmed using a general-purpose programming language and a low-level graphical library. Figure 23 portrays the layout model.

3.3. Transformation Rules

Model transformation is the process of converting one or more models – called source models – to an output model – the target model – of the same system. Transformations may combine elements of different source models in order to build a target model. Transformation rules apply to all the types of models listed above.

The following steps make up the list of transformation rules suggested in [17], and are considered as part of our framework:

1. Maintain tracking structures of all class instances where needed
2. Maintain tracking structures for Association populations where needed
3. Support state-machine semantics
4. Enforce Event ordering
5. Preserve Action atomicity
6. Provide a mapping for all analysis elements, including:
   - Domain, Domain Service
   - Class, Attribute, Association, Inheritance, Associative Class, Class Service
   - State, Event, Transition, Superstate, Substate
   - All Action-modeling elements

   The transformations between models [8] provide a path which enables the automated implementation of a system to be derived from the various models defined for it.
4. POMA Architecture

4.1 Overview of POMA

The proposed architecture consists of six architectural levels of models using patterns (Figure 18) and UML specifications of system architecture for interactive systems development called POMA (Pattern-Oriented and Model-Driven Architecture). POMA enables us to specify and build interactive systems. It is an architecture which supports both novices and experts in system development.

The architecture of system development (Figure 18) includes:
- A library of patterns by category used in the architecture with their selection, composition, and mapping;
- The various models that we propose for the development of interactive systems with their transformation (PIM to PIM, and PSM to PSM) and their mappings which are: PIM to PSM.

POMA is based on:
- Six architectural levels and categories of patterns;
- Ten models, five of which are [POMA.PIM] and five others which are [POMA.PSM];
- Four types of relations used in POMA architecture, which are:
  1. Composition: used to combine different patterns to produce a [POMA.PIM] by applying the composition rules
  2. Mapping: used to build a [POMA.PIM] which becomes a [POMA.PSM] by applying the mapping rules
  3. Transformation: used to establish the relationship between two models (PIM or PSM) by applying the transformation rules
  4. Generation: is used to generate the source code of the whole system by applying the generation code rules

The direction in which to read the POMA architecture in Figure 18 is as follows:
- Vertically, it is about the composition of the patterns to produce ten PIM and PSM models.
- Horizontally, it is about the composition and mapping of the patterns to produce five PIM and five PSM models, and the generation of the source code of the whole system (not included in this research).
Figure 18: POMA architecture for interactive systems development
5. Illustrative case study

This section presents a case study which describes the design of a non-functional UI prototype of a simplified environmental management system (Figure 23), illustrating and clarifying the core ideas of our approach and its practical relevance. It can also be used by an object-oriented designer to learn how to use design patterns.

This environmental management system allows the analysis of the environment, its evolution and its economic and social dimensions, and proposes some indicators of performance. The main objectives of environmental management are the treatment and distribution of water, improving air quality, monitoring noise, the treatment of waste, monitoring the health of fauna and flora, monitoring land use, preserving coastal and marine environments, and managing natural and technological risks (Figure 23).

Note that only a simplified version of the environmental management system will be developed here. The system and corresponding models will not be tailored to different platform or user roles. The main purpose of the example is to show that models consist of a series of model transformations, in which mappings from abstract to concrete models must be specified. In addition, it illustrates how patterns are used to establish the various models, as well as to transform one model into another while respecting the patterns composition rules described above and the patterns mapping rules.

We present below a general overview of the PIM and PSM models of the environmental management system in applying the pattern composition steps and mapping rules, as well as the transformation rules for these five models. The details of this illustrative case study are presented in the Appendix, in which the following five models representing the same system are illustrated on a laptop platform and on a PDA platform: Domain model, Task model, Dialog model, Presentation model, and Layout model of the POMA architecture. Table 2 lists the patterns that will be used by the system.

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Model Type</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>Domain</td>
<td>The user’s identity needs to be authenticated in order to be allowed access to protected data and/or to perform authorized operations.</td>
</tr>
<tr>
<td>Multi-Value Input Form</td>
<td>Domain</td>
<td>The user needs to enter a number of related values. These values can be of different data types, such as “date”, “string”, or “real”.</td>
</tr>
<tr>
<td>Submit</td>
<td>Domain</td>
<td>The user needs to submit coordinates to the authentication process to access the system.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Domain</td>
<td>The user needs help concerning the use of the Login Form.</td>
</tr>
<tr>
<td>Close</td>
<td>Domain</td>
<td>The need to close the system from the Login form</td>
</tr>
<tr>
<td>Find (Search, Browse, Executive Summary)</td>
<td>Task</td>
<td>The need to find indicators related to the task concerned, to find environmental patterns related to the indicators, and to find a presentation tool to display the results of the indicators and the environmental patterns</td>
</tr>
<tr>
<td>Path (Breadcrumb)</td>
<td>Task</td>
<td>The need to construct and display the path that combines the data source, task, and/or subtask</td>
</tr>
<tr>
<td>Index Browsing</td>
<td>Task</td>
<td>The need to display all indicators listed as index</td>
</tr>
<tr>
<td>Pattern</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Adapter</td>
<td>Task</td>
<td>The need to convert the interface of a class into another interface that clients expect; an adapter lets classes work together which could not otherwise do so because of interface incompatibility</td>
</tr>
<tr>
<td>Builder</td>
<td>Task</td>
<td>The need to separate the construction of a complex object from its representation, so that the same construction process can create different representations</td>
</tr>
<tr>
<td>List</td>
<td>Task</td>
<td>The need to display the information using forms</td>
</tr>
<tr>
<td>Table</td>
<td>Task</td>
<td>The need to display the information in tables</td>
</tr>
<tr>
<td>Map</td>
<td>Task</td>
<td>The need to display the information in geographic maps</td>
</tr>
<tr>
<td>Graph</td>
<td>Task</td>
<td>The need to display the information in graphs</td>
</tr>
<tr>
<td>Home Page</td>
<td>Task</td>
<td>The need to define the layout of an interactive system home page, which is important because the home page is the interactive system interface with the world and the starting point for most user visits</td>
</tr>
<tr>
<td>Wizard (Welie, 2004)</td>
<td>Dialog</td>
<td>The user wants to achieve a single goal, but several decisions and actions need to be taken consecutively before the goal can be achieved.</td>
</tr>
<tr>
<td>Recursive Activation (Paternò, 2000)</td>
<td>Dialog</td>
<td>The user wants to activate and manipulate several instances of a dialog view.</td>
</tr>
<tr>
<td>Unambiguous Format</td>
<td>Presentation</td>
<td>The user needs to enter data, but may be unfamiliar with the structure of the information and/or its syntax.</td>
</tr>
<tr>
<td>Form</td>
<td>Presentation</td>
<td>The user must provide structural textual information to the system. The data to be provided are logically related.</td>
</tr>
<tr>
<td>House Style (Tidwell, 2004)</td>
<td>Layout</td>
<td>Usually, the system consists of several pages/windows. The user should have the impression that it all “hangs together” and looks like one entity.</td>
</tr>
</tbody>
</table>

Table 2: Pattern Summary

To visualize the models, we have used an extended version of the ConcurTaskTree (CTT) notation. In addition to the predefined types, we have enhanced the CTT by adding a fifth type: The Pattern Task. Figure 19 shows the graphical representation of the pattern.

Figure 19: Pattern symbol
The [POMA.PIM]-independent model of the environmental management system is obtained by composing patterns between them and by taking into account the patterns composition rules – see Figure 20.
The mapping rules of the patterns of the environmental management system for laptop and PDA platforms are listed in Table 3.

<table>
<thead>
<tr>
<th>HCI patterns of the Microsoft platform</th>
<th>Type of mapping</th>
<th>Replacement patterns for the laptop platform</th>
<th>Replacement patterns for the PDA platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1. Login</td>
<td>Identical</td>
<td>P1. Login</td>
<td>P1. Login</td>
</tr>
<tr>
<td>P3. Submit</td>
<td>Scalable or fundamental</td>
<td>P3. Submit</td>
<td>P3.s. Submit (smaller button)</td>
</tr>
<tr>
<td>P6. Find (Search, Browse, Executive Summary)</td>
<td>Identical, Scalable</td>
<td>P6. Find (Search, Browse, Executive Summary)</td>
<td>P6. Find (Search, Browse, Executive Summary)</td>
</tr>
<tr>
<td>P7. Path (Breadcrumb)</td>
<td>Identical, Scalable (Laptop) -Scalable or fundamental (PDA)</td>
<td>P7. Path (Breadcrumb)</td>
<td>- P7.1.s. Shorter Breadcrumb Trial - P7.2. Drop-down “History” menu</td>
</tr>
<tr>
<td>P8. Index Browsing</td>
<td>Identical</td>
<td>P8. Index Browsing</td>
<td>P8. Drop-down menu</td>
</tr>
<tr>
<td>P12. Table</td>
<td>Identical</td>
<td>P12. Table</td>
<td>P12. Table</td>
</tr>
<tr>
<td>P14. Graph</td>
<td>Identical</td>
<td>P14. Graph</td>
<td>P14. Graph</td>
</tr>
</tbody>
</table>

Table 3: Example of pattern mapping of the of the environmental management system model for laptop and PDA platforms
After the mapping, we obtain the PSM Model of the environmental management system for the laptop platform – see Figure 21 for the UML diagram of the PSM environmental management system model for the laptop platform.

Figure 21: UML diagram of the PSM environmental management system model for the laptop platform
After the mapping, we obtain the PSM environmental management system model for the PDA platform – see the UML diagram of the PSM environmental management system model for the PDA platform in Figure 22.
A final UI of the environmental management system is shown in Figure 23.

![Figure 23: Screenshot of the environmental management system for the laptop platform](image)

6. Conclusion

In this paper, our discussion focused on an architectural model combining two key approaches: model-driven and pattern-oriented. We first describe an architectural level and categories of patterns (Navigation patterns, Interaction patterns, Visualization patterns, Presentation patterns, Interoperability patterns, and Information patterns), as well as the various relationships between patterns. These relationships are used to combine and map several types of patterns to create a pattern-oriented design for interactive systems, as well as to show how to generate specific implementations suitable for different platforms from the same pattern-oriented design. Then, we proposed five categories of models (Domain model, Task model, Dialog model, Presentation model, and Layout model) to address some of the challenging problems, such as: (a) decoupling the various aspects of interactive systems, such as business logic, UI, navigation, and information architecture; and (b) isolating platform-specific problems from the concerns common to all interactive systems. Finally, we presented a case study to illustrate and clarify the core ideas of our approach and its practical relevance.

The Model-View-Controller (MVC), Model-View-Presenter (MVP), Presentation-Abstraction-Control (PAC), Seeheim, Arch/Slinky, PAC-Amadeus, and POMA architectures are similar in many ways, but each has evolved to address a slightly different concern. By becoming familiar
with these architectures and other related architecture models, developers and architects will be better equipped to choose an appropriate solution in their next design endeavor, or possibly in the creation of future pattern-oriented and model-driven architectures. The current limitations of the POMA are the following:

- There is a need to define measures to study the usability patterns that could be used in the POMA.
- The patterns do not allow flexibility between the platform-independent specification of interfaces, the platform-specific form of those interfaces, and the eventual implementation of those interfaces.
- The patterns do not constitute an approach to the full life cycle integration and interoperability of enterprise systems comprising software, hardware, humans, and business practices.
- The POMA does not encourage the designer to consider other aspects of the dialog which are very important to the user, like the help function or error-handling.
- The POMA does not facilitate the use of the design constraints or the description of the interface, which are of great importance to the designer [57; 58; 59; 60].
- Patterns are signs of weakness in programming languages.
- Finding and applying the appropriate architectural patterns in practice still remains largely an ad hoc and unsystematic process, e.g. there is a lack of consensus in the community with respect to the “philosophy” and granularity of architectural patterns, and a lack of a coherent pattern language.

Further research is required to address these limitations, one by one.

The strengths of the POMA architecture are the following:

- The POMA facilitates the use of patterns by beginners as well as experts
- The POMA supports the automation of both the pattern-driven and model-driven approaches to design.
- The POMA ensures the quality of the applications produced, since a pattern-oriented architecture will also have to enable the encapsulation of quality attributes and to facilitate prediction.
- The POMA supports the communication and reuse of individual expertise as regards good design practices.
- The POMA integrates all the various new technologies (including, but not limited to, traditional office desktops, laptops, palmtops, PDAs with and without keyboards, mobile telephones, and interactive televisions).

In terms of the evolution of the POMA architecture, some of the next steps will include:

- Standardization of the POMA architecture to all types of applications, and not only interactive systems;
- Proposal of a process for the generation of source code from five POMA PSM models;
- Building of a tool to allow the automation of the entire POMA architecture process to facilitate the development of interactive applications of any category for both novice and expert users.
7. Acknowledgments

We extend special thanks to Olivier ALNET, who participated in the development of the POMA architecture example.

8. References


