DPI: A Conceptual Model Based on Documents and Interaction Instruments

Olivier Beaudoux[†] & Michel Beaudouin-Lafon[‡]

[†] *Département Génie Informatique, Réseaux et Télécoms, Ecole Supérieure d'Electronique de l'Ouest, 4 rue Merlet de la Boulaye, 49009 Angers, France*

Tel: *+33 2 41 86 67 67*
Fax: *+33 2 41 87 99 27*
Email: *olivier.beaudoux@eseo.fr*
URL: *http://www.lri.fr/~beaudoux*

[†] *Laboratoire de Recherche en Informatique, Université Paris-Sud, Bâtiment 490 - 91405 Orsay, France*

Tel: *+33 1 69 15 69 10*
Fax: *+33 1 69 15 65 86*
Email: *mbl@lri.fr*
URL: *http://www.lri.fr/~mbl*

The DPI model (Documents, Presentations, Instruments) offers an alternative to current application-centered environments by introducing a conceptual model based on documents and interaction instruments. DPI makes it possible to edit a document through multiple simultaneous presentations. The same instrument can edit different types of content, facilitating interaction and reducing the user's cognitive load. DPI includes a functional model, aimed at the user interface designer, that describes implementation principles in terms of properties, services and representations. The DPI model offers a first but essential stage in designing and implementing a new generation of document-centered environments based on a new interaction paradigm.

Keywords: conceptual model, document-centered interaction, instrumental interaction, compound document, interactive workspace, desktop environment, metaphor, action, perception.

1 Introduction

Most current desktop environments are based on applications, each dedicated to handling a particular type of data, such as text, image or vector-based drawing. Windows were introduced to facilitate switching among applications and techniques such as copy-paste were created to enable the transfer of contents from one application to another. Unfortunately, many tasks require multiple applications. A user may be forced to juggle four or five applications in order to create a single web page or technical document. Software vendors use three strategies to reduce the resulting complexity and higher cognitive load.

The first creates "mini-applications" within a larger application. For example, Microsoft Office has three applications (Word, PowerPoint and Excel) that each include functions for vector-based drawing. Not only do these offer significantly fewer functions than a full-scale vector-drawing application, which is still needed for complex drawings, but the user interfaces are somewhat different and the data formats are incompatible. This approaches duplicates functionality without solving the problem.

Open architectures offer another approach, allowing third parties to develop and market extensions called *plug-ins*. For example, Adobe Photoshop (Gray, 1997) offers a wide variety of plug-ins to edit images, including creation of vector-based drawings and simple 3D models. Other image-editing applications are compatible with Adobe Photoshop's plug-ins so software vendors who develop Photoshop plug-ins have a larger market. The market of plug-ins for QuarkXPress (page layout) and Macromedia Director (multimedia authoring) is also very active, and some plug-ins are more expensive than the parent application. Users install plug-ins to extend and specialise applications according to their needs, and can use the same plug-in in different applications. However, the user interfaces of plug-ins are often poorly integrated into the parent application and are accessible only through bulky dialog boxes.

The third approach explicitly produces multiple applications with compatible user interfaces. The best example is the Adobe suite: Photoshop (image edition), Illustrator (drawing), GoLive (Web site design) and InDesign (page layout) have the same visual presentation of the interface and similar tool palettes. All support layered documents, and layers created in one application can be imported into another application. Sometimes the link can be "live": the imported layer is automatically updated when it is edited in its original application. However, this approach does not relieve users from dealing with multiple documents for a single task: For a live link to work, the document holding the imported layer must be kept on disk. In addition, applications that do not belong to the suite cannot take advantage of the integration of functions within the suite.

The goal of all these approaches is to put the document, rather than the application, at the center of the interaction. Unfortunately they all fail because

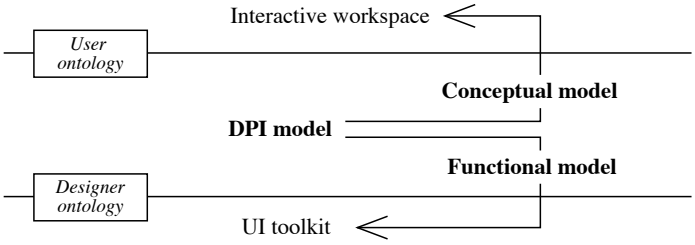

Figure 1: The two levels of the DPI model.

users must still juggle multiple applications and/or multiple documents for a single task. They try to make applications less visible by blurring the boundaries between applications, but the underlying logic is still application-centric. Historically, the Xerox Star (Johnson et al., 1989) was the first major system to adopt a document-centered approach. More recently, frameworks such as OpenDoc (Apple, 1994) and OLE (Brockschmidt, 1995) have made applications less visible by supporting *compound documents*: a document is not managed by an application dedicated to its type, but each of its parts is handled by a specialised application or *part editor*. These approaches do not discard applications but make them less visible. For example, when a part of a document is clicked, menu bars and tool palettes are reconfigured to allow editing its content. This is similar to switching between applications. The main advantage is the *in-place* editing that saves copy-pasting between applications. But the transition from one document part to the next creates a break in the interaction since each part manages its own interface: there is no sharing of tools among part editors.

In summary, even though systems are becoming more document-centric, the user interfaces are still application-centric. The goal of this paper is to explore the problem of document-centered interaction. We propose the Documents–Presentations–Interactions (DPI) model, a conceptual model that separates document *contents* from document *interaction*.

The DPI model combines a document model compatible with XML (W3C, 2000a) and an interaction model based on instrumental interaction (Beaudouin-Lafon, 1997). We present two levels of the DPI model: the conceptual model that matches a *user ontology*, and the functional model that matches a *designer ontology* (Figure 1). A concrete implementation of the DPI model must define an interface model that matches the conceptual model and a software architecture that supports the functional model. Our longer term goal is to create a toolkit for building a new generation of document-centered environments.

The next two sections describe the conceptual and functional models, respectively. We then compare DPI with related work and conclude with a discussion of future work.

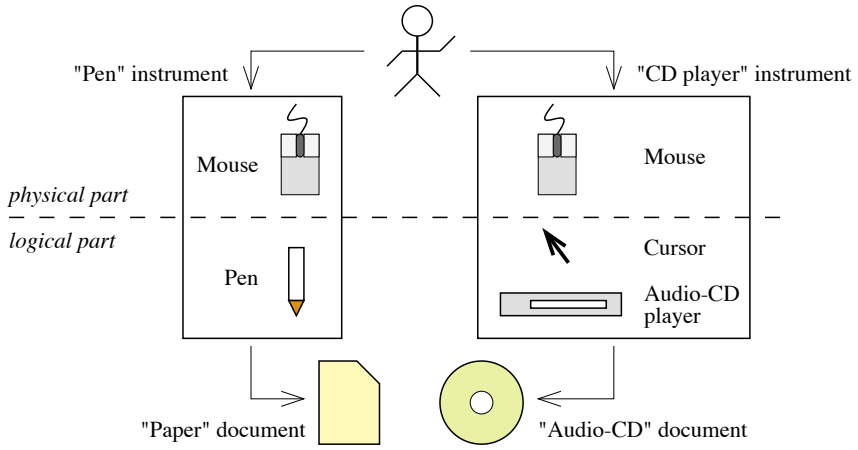

Figure 2: Instrument description

2 Conceptual Model

In a user ontology* the concepts of document and instrument are natural: the document is a data repository and the instrument is a means for creating and modifying documents. Therefore the DPI model is based on the document metaphor and the instrument metaphor.

2.1 The Instrument Metaphor

In daily life, we often use tools or instruments to operate on physical objects. For example, we use a pen to write on a sheet of paper. This observation forms the basis of the instrumental interaction model (Beaudouin-Lafon, 1997; Beaudouin-Lafon, 2000). According to this model, an instrument mediates between a user and *target object* (or object of interest).

An instrument has two facets: physical and logical. The physical facet exists outside the system. It includes input-output transducers used by the instrument. Input transducers capture the physical actions of the user and output transducers provide feedback information. The logical facet exists inside the system and its state is made perceivable outside the system. It includes methods for transforming user actions onto the logical instrument (input), and the representation of the instrument itself (output).

Instruments can be more or less direct. For example, using a pen to write on paper is more direct than using a CD player to play a CD. A similar indirection occurs when the pen or the CD player is simulated by an interactive system and is operated with a mouse (Figure 2).

*The term *ontology* here means a description of the concepts and relationships that are meaningful to a subject (in our case, the users and designers of interactive systems).

Most instruments modify existing objects. However instruments can also be used to change the *perception* of an object, as with magnifying glasses (Perlin & Fox, 1993) and Magic Lenses (Bier et al., 1993). Such instruments create *alternate presentations* of documents (see next section). In general, these instruments are direct but the manipulation affects the way the instrument operates rather than the target object.

Interaction instruments can be organised into three categories:

- *Direct instruments*: The users acts *directly* on the document. For example, clicking the mouse on an object and moving the hand moves the object. The action of the hand on the instrument and its effect on the target object are perceived as similar.

- *Indirect instruments*: The user acts *directly* on the instrument but *indirectly* on the document. For example, entering a number in a dialog box changes the font size of the selected text. The action of the hand on the instrument and its effect on the target object are perceived as related but different.

- *Perception instruments*: The user acts *directly* on the instrument but *not* on the document. For example, moving the thumb of a scrollbar changes the part of the document that is being viewed. The action of the hand on the instrument affects the presentation of the target object, not its contents.

2.2 The Document Metaphor

In the physical world, documents such as books have two facets:

- *Persistence*: the document provides on-going support for its contents, e.g. the paper absorbs the ink. When writing on a document, the user perceives the persistent result of his actions. When reading a document, the user perceives what has been made persistent and then interprets it.

- *Presentation*: the document has a concrete appearance. The appearance is intrinsic to the document: a document has a single presentation which is a direct result of the persistent information it contains.

Persistence and presentation are naturally coupled in physical documents. However they are decoupled in electronic documents. The file system manages persistence, and output devices, such as screens, manage presentation. This decoupling is evident when a document that appears on the screen must be explicitly saved onto a persistent storage medium.

An advantage of this decoupling is the ability to view several presentations of a document at the same time, e.g. an outline or page layout presentation of a text document, or a visual or audio presentation of a musical partition. While this is useful from the user's perspective, it breaks the physical document metaphor. In the user ontology, we want to hide the decoupling between document and presentation as much as possible.

2.3 Multiple Presentations Abstraction

If each document had only one presentation, the notion of presentation would be unnecessary in a user ontology. However, the advantages of multiple presentations are well-known (see the Zelig system, Celentano et al. (1992), for an example). So our goal is to follow the physical document metaphor while supporting multiple presentations: from the users' point of view, any presentation of the document should be the document itself.

We could introduce multiple presentations by using the cameras and monitors metaphor from X_{TV} (Beaudouin-Lafon et al., 1990): a document is filmed by one or more cameras and can be visualised by the equivalent of video monitors. This metaphor is easy to understand. Users may edit a document with its *natural* presentation, and visualise it through *alternate* presentations provided by the monitors. This approach preserves the initial document metaphor, but goes against the principle of direct manipulation (Shneiderman, 1983) since the alternate presentations are passive.

The DPI model extends this metaphor by making alternate presentations active, i.e. the document can be edited through *any* presentation. This introduces a fundamental requirement: editing results must be *synchronous* across presentations. This ensures that the abstract idea of multiple presentations becomes concrete to users. They observe that multiple presentations are updated synchronously and are therefore the *same* document.

Multiple editable presentations also open the way to *shared editing*, i.e. supporting simultaneous editing of the same document by multiple users from their respective workstations. This groupware extension to DPI is beyond the scope of this article.

2.4 Compatibility between Model Components

The DPI conceptual model describes how users manage documents by using instruments on their presentations and how they perceive the results of their actions. Based on Norman's action theory (Norman & Draper, 1986), the diagram in Figure 3 presents the model as an action and perception flow diagram with three levels: user, instrument and document.

- The *user level* states that users specify their intentions through actions and interpret the results using their senses.

- The *instrument level* describes how instruments transform user actions. Instruments support three main functions: navigation, perception and editing. Navigation can be carried out by an indirect instrument such as a search instrument, by a direct instrument such as a scrollbar or by a perception instrument such as a radar view. Editing can be carried out by an indirect instrument such as a spell checker or by a direct instrument such as a pen. Perception is carried out either directly by the natural document presentation, or indirectly by perception instruments that provide alternate presentations, such as a magnifying glass.

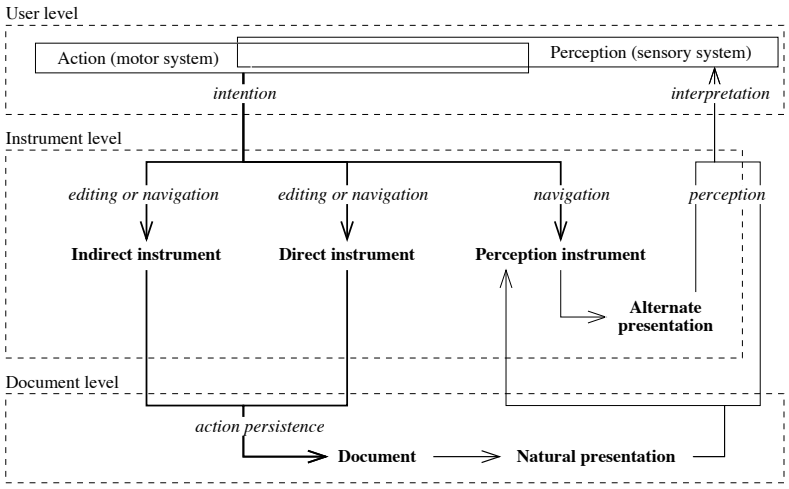

Figure 3: DPI conceptual model as an action-perception diagram. Thick lines represent the action flow and thin lines represent the perception flow. For clarity, the perception of instruments is omitted.

- The *document level* reveals the double role of documents: persistence of user actions and presentation of contents.

According to the ecological theory of perception (Gibson, 1979), action and perception are strongly coupled: the user must perceive before acting, e.g. by locating an object before selecting it, and must act in order to perceive, e.g. by navigating through a document in order to locate an object. This is supported in DPI by the three functions of instruments, editing, navigation and perception. However an effective coupling between action and perception requires additional *compatibility rules* among the three levels of the model:

1. *User* $\leftrightarrow$ *instrument* compatibility: interaction with an instrument should match its function. For example, a brush-like instrument should be used to change an object's color. This relates to the concept of *affordance* (Gibson, 1977): instruments must express their functions in a directly-perceivable way.

2. *Instrument* $\leftrightarrow$ *document* compatibility: an instrument works with specific types of documents. For example, a pen is adapted to a sheet of paper because paper can receive ink. This compatibility defines the possible interactions between instruments and documents.

3. *Document* $\leftrightarrow$ *user* compatibility: the natural presentation of a document should be suited to our senses. For example, a sheet of paper is compatible with visual perception, while a Braille display is compatible with tactile (and possibly visual) perception.

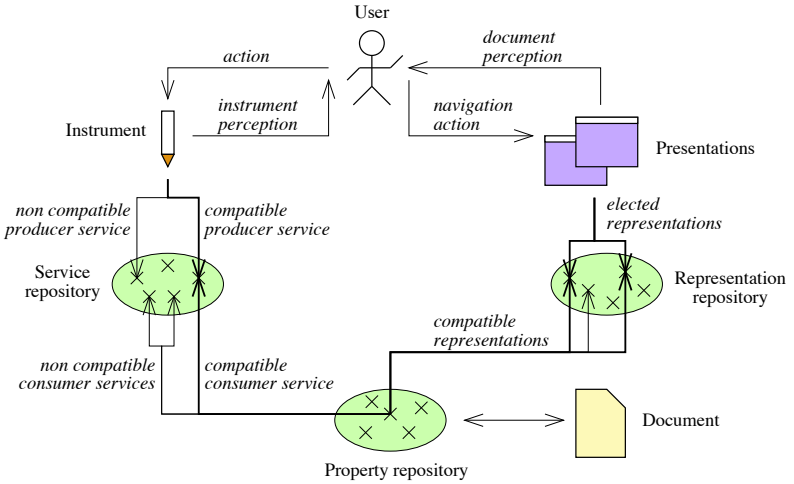

Figure 4: DPI functional model. Thick lines represent a sample instrument to presentation interaction path. Thin lines represent other possible interaction paths.

We take compatibility in physical documents for granted; we are not aware of compatibility *per se* but rather the lack of it. The goal of the DPI conceptual model is to generate a comparable set of possible combinations among electronic documents, instruments and presentations. By separating instruments from documents, the DPI model supports instruments that can act on documents of different types, increasing the number of possible combinations. These combinations should simplify interaction and reduce the users' cognitive load while offering a rich set of functions. For example, the same instrument could be used to change the color of a title in a text document and the color of an arrow in a drawing. Current desktop environments do not support these kinds of facilities.

3 Functional Model

The functional model transcribes the conceptual model for the interface designer. We express it in a designer ontology by reifying the conceptual model's relationships into primitives (Figure 4), described in sections Sections 3.1–3.3. We then combine these into documents and presentations (Section 3.4) as well as devices and instruments (Section 3.5).

3.1 Persistence Primitive: The Property

A document and its presentations are defined by their *properties*. A property *definition* is a pair *(name, type)* and a property *value* is a pair *(name, value)*. A property can be *atomic* (it defines a single value), *composite* (it defines a value composed of a predefined set of properties), or a *set* (it defines a value composed of an arbitrary set of properties).

Name-value associations are defined in documents and form their states. *Name-type* associations define the properties' *schema*. Document properties are visualised through presentations and edited using instruments. Some properties cannot be edited, such as a document's creation date. This information, i.e. that it cannot be edited, is stored in the schema.

The properties in the DPI document model provide fine-grained access to the contents of a document. Interaction instruments can act on specific properties, independently of document types. Moreover, the document format is open and extensible because the properties' schemas are available and extensible. For example, the document format could be based on XML (W3C, 2000a) and the property schemas on DTDs or XML schemas. In contrast, OpenDoc or OLE provide coarse-grained access at the level of *parts*[†], and parts are stored in a proprietary format.

3.2 Action Primitive: The Service

All objects involved in the interaction process are connected through *services*. A service is the means by which user actions are transmitted along the action-perception chain. *Producer services* can activate *consumer services* in a chain from input devices to document properties via instruments. The *activation* of a service is based on the instrument $\rightarrow$ document compatibility described in the conceptual model. Activation of consumer service S_{in} in consumer C by producer service S_{out} in producer P is defined as follows:

1. P and C are *likely to interact* if P is a direct instrument geometrically pointing at C, or if P is an indirect instrument targeting C.

2. If P and C are likely to interact, a *compatibility* test is carried out between each consumer service $(S_{out,i})_{i=1..n}$ of P and each producer service $(S_{in,j})_{j=1..p}$ of C. A *connection* $S_{out,i} \rightarrow S_{in,j}$ is made for each compatible pair $(S_{out,i}, S_{in,j})$.

3. A connection $S_{out,i} \rightarrow S_{in,j}$ is *activated* when the service $S_{out,i}$ is invoked. The producer must choose the invoked service if there is an ambiguity. For example, an instrument that can move and resize an object provides two services. Once the two services are connected, this instrument provides the user with a means of choosing which one to activate, for example by offering two activation buttons.

4. Services remain connected as long as P and C are likely to interact.

The simplest compatibility test between services is the equality of their names. However, it is useful to define more generic compatibility rules. For example, services can be organised in a tree so that two services S_1 and S_2 are compatible if and only if S_1 is an ancestor of S_2. An instrument that provides the generic `move` service would then be compatible with properties that provide specialised services such as `move_icon` or `move_note` if `move_note` and `move_icon` were descendants of `move` in the service tree.

[†] A document is composed of parts, each with a specified data type such as formatted text or vector-based drawing.

Services reify the traditional notion of a command: a service is a command that an instrument can apply to a target document property. Generic (or polymorphic) instruments directly result from the compatibility rule: the same instrument can edit different kinds of properties as long as it is compatible with them. Thus, the concept of service embodies the reification and polymorphism principles of Beaudouin-Lafon & Mackay (2000). From an interaction point of view, instrument services define a fine-grained interaction, at the instrument and property levels, while OpenDoc part editors (Apple, 1994), for example, define a coarse-grained interaction at the editor and part levels.

3.3 Perception Primitive: The Representation

Properties are perceived by users through *representations*, *elected* among those compatible with the property. Some standard (or canonical) representations are provided for simple types (e.g. integer, character string), composite types (e.g. date, time) and set types (e.g. trees, lists, icons). Perception instruments can transform properties of a document-specific schema into properties of a well-known schema with standard representations.

Electing representations works like activating services: a representation provided by a document is selected if it is compatible with the presentation medium. In order to create a presentation, a representation is elected either automatically or with user assistance. This election mechanism corresponds to the document $\rightarrow$ user compatibility of the conceptual model.

As for services, the use of representations allows a fine-grained control of the document presentation, at the property level, whereas in OpenDoc, part editors are coarse-grained, at the part level.

3.4 Documents and Presentations

The DPI model does not make any assumption about the organisation of documents. The model of Dourish et al. (1999) is a better candidate than a traditional hierarchical file system because it uses collections and dynamic queries and it is based, like DPI, on document properties (see Beaudoux (2000) for more details). Documents can also be used to represent workspaces, collections, and even instruments. This homogeneity should help users organise their environment. For example, documents could be queried by specialised languages such as XPath (W3C, 1999) and XML Query (W3C, 2000c).

A document is a tree of property values and a presentation is a tree of representation property values. For graphical interaction, these representations include geometric properties such as location and size. A direct instrument targets a document property through its presentation, therefore representations must maintain a link back to the properties they represent. Moreover, presentations may contain properties that do not correspond to any document property. For example, in a representation using icons, the positions of icons are unlikely to correspond to properties of the underlying document. Thus, presentations must store their own properties with their associated documents. This means that a style sheet approach such as XSL (W3C, 2000b) cannot be used to transform a document into a presentation: a more powerful mechanism is necessary.

Multiple presentations make it possible to edit a document according to several perspectives. Consider a scenario involving the creation of material for a course, composed of handouts for the students, slides for the presentation and notes for the teacher. The natural presentation of the document is an outline that facilitates navigation through the course hierarchy and editing at the semantic level. At the layout level, each facet of the course (handouts, slides and notes) has a dedicated alternate presentation with appropriate formatting of figures and text. These four presentations make it easy to edit various aspects of the course in parallel.

Representations are defined in a *repository* that changes dynamically. When a document is (down)loaded into the system, associated presentations may have elected representations that are not defined in this repository. In this case, the necessary representation definitions can be (down)loaded as well. This is similar to loading plug-ins into a Web browser.

3.5 Devices and Instruments

3.5.1 Physical, Logical and Simulated Devices

Input devices provide producer services and are defined as a tree of sensors — see Beaudoux (2000) for more details. For example, a one-button mouse has three sensors: two potentiometers and one button, and produces three services: single-click, double-click and drag-and-drop. In order to support a wide variety of platforms and input devices, the DPI model defines physical, logical and simulated devices.

A *physical device* corresponds to a hardware peripheral such as the above one-button mouse.

A *logical device* is built from one or more devices. For example, a logical three-button mouse can be built by combining the physical one-button mouse with two keyboard keys used as additional button sensors. This follows Myers (1989) *Interactors* model.

A *simulated device* emulates a physical device. For example, the Xerox Star (Johnson et al., 1989) displays soft keyboards on the screen to input special characters such as mathematical formulas with the mouse. X_{TV} (Beaudouin-Lafon et al., 1990) supports the creation of arbitrary simulated devices. *Metamouse* (Maulsby et al., 1989) uses a simulated mouse to demonstrate procedural actions.

3.5.2 Functional Model of Instruments

An instrument receives the user actions from an input device and transforms them into commands sent to the target property of a document (Figure 5). The *physical part* of the instrument defines its consumer services, i.e. what input it accepts. For example, a `move` instrument may accept two services: `drag-and-drop` and `constrained drag-and-drop`. The instrument may use an existing device or create a dedicated logical device as described in the previous section. For example, the instrument may create a logical device that combines the one-button mouse with the keyboard's shift key in order to provide both `drag-and-drop` and `constrained drag-and-drop` services. When multiple devices are available, the instrument may ask the user or use a document that specifies priorities and preferences. Once the device is selected, the producer services of the elected input device are connected to the consumer services of the instrument (1).

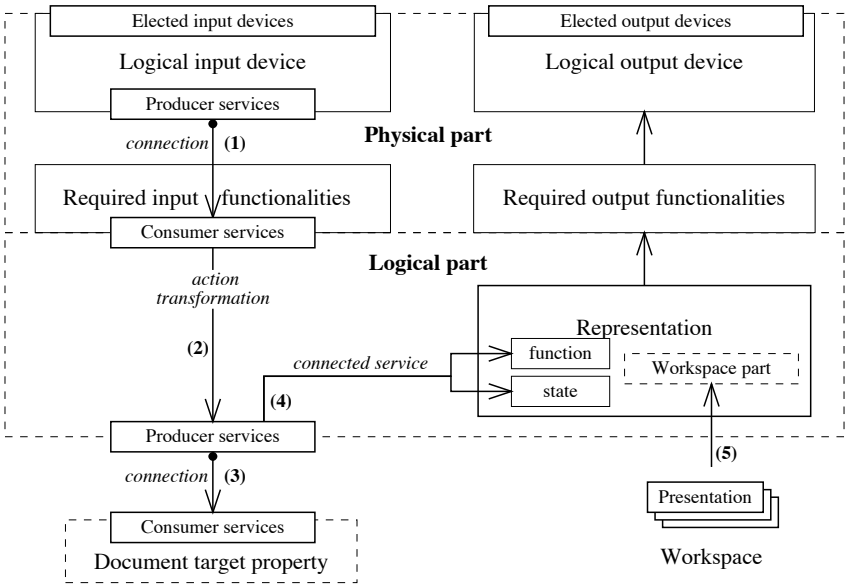

Figure 5: Functional model of an instrument

The *logical part* of the instrument defines its producer services. For example, the `move` instrument provides the `change-position` service. When an instrument's consumer service is activated by a user action on a device, it is transformed into a producer service (2), which in turn activates a consumer service of the document (3). For example, a `drag-and-drop` action on the mouse is transformed into a `change-position` command applied to the target property of the document.

Instruments must themselves be represented on output devices. This representation should include the function and state of the instrument (4). Perception instruments must also manage the alternate representation of their target documents (5), e.g. magnifying glasses must generate magnified presentations of the documents underneath them.

The persistence of instrument properties is managed transparently by an associated document. This document contains the instrument state, configuration (logical device, behavior, representation), and online help. Sleeter (1996) discusses the importance of online help in document-centered systems. Online help tends to be centralised in traditional environments while document-centered systems must provide help at different levels: workspace, documents and instruments.

The generic aspect of instruments with respect to documents results from the generic aspect of services with respect to properties (Section 3.2). The same instrument can be used in several contexts. For example, a pen can be used to enter text in any writable document, to write annotations in any document, and to

edit the label of a workspace element, e.g. the name of a document. The service compatibility between instruments and document properties results in the ability to dynamically create contextual palettes of instruments. Traditional contextual menus are predefined by the application and depend only on their target object. The DPI model makes it possible to dynamically compute the palette of activatible instruments for a given set of properties.

Finally, instruments can be *chained* together. For example, combining a pen (direct instrument) with a magnifying glass (perception instrument) permits precision drawing as the pen operates *through* the magnifying glass, as with Magic Lenses (Bier et al., 1993). To our knowledge, no other general system supports such facilities.

Instruments, like representations, are declared in a *repository* that changes dynamically. When a document is (down)loaded into the workspace, related instruments can also be (down)loaded. This offers greater flexibility than plug-ins.

4 Related Work

4.1 Document Models

OpenDoc (Apple, 1994) was the most advanced attempt at creating a document-centered system, although it was developed in the context of an existing application-centered environment (MacOS). Dykstra-Erickson & Curbow (1997) explain that users do not have to interact with applications in order to edit documents: they interact with the documents themselves. The document model is based on the metaphors of paper and *parts*. A document is a set of parts, and each part can be handled by a specialised *editor*. In our approach, documents are *not* the single most important entity in the system: instruments are the main interaction component, while documents handle persistence and are the targets of instruments. The main differences between OpenDoc and DPI can be summarised as follows:

- OpenDoc editors have a finer grain than traditional applications, but a coarser grain than DPI instruments. Similarly, OpenDoc parts have a coarser grain than DPI document properties.
- OpenDoc editors do not separate interaction from display whereas DPI instruments are decoupled from presentations.
- OpenDoc editors are specialised for a given part type while DPI instruments can be generic and DPI presentations can be canonical.

DPI supports a more seamless style of interaction than OpenDoc: in OpenDoc, the interactive environment, e.g. menu bars and palettes, changes when navigating from part to part, whereas a single DPI instrument can be used across all documents in the workspace.

The OLE framework (Brockschmidt, 1995) uses inter-application communication to allow multiple applications to operate on the same document. OLE is not a document-oriented system, but does make it possible to edit compound documents. Within the Microsoft Office suite, applications have homogeneous

interfaces and the interaction through OLE is relatively seamless. However this continuity breaks down when using other applications.

OOE (Backlund, 1997) is a NextStep system extension that manages composite documents in a simple way: clicking a part invokes its creator application. The display of compound documents takes advantage of *display PostScript*. Since in-place part editing is not available, OOE is in fact application-centered.

HotDoc (Buchner, 2000) is a Smalltalk extension to the well-known MVC design pattern. It supports compound documents by defining the `PartApp`, `PartView` and `PartController` classes. Unlike OLE and OOE, HotDoc does support multiple views (or presentations) of the same model (or document). HotDoc is based on a part-based model similar to OpenDoc, with similar drawbacks.

4.2 Interaction Models

Myers (1989) *Interactors* encapsulate interaction in a small number (seven) of object types. An interactor is not based on an input device type but on the *nature* of the interaction, e.g. defining a point or selecting an item in a list. This approach emphasises the separation between interaction and visualisation and the ability to use various devices for the same task.

Our contribution consists of separating devices from instruments and supporting physical, logical and simulated devices. Logical and simulated devices are transparent to the user and appear only in the designer ontology. Also, DPI is not limited to seven basic devices or instruments. In fact, instruments can be strongly typed or highly generic, according to their function.

The instrumental interaction model is the foundation of the DPI model, but does not constitute a conceptual model. By combining the concepts of interaction instruments and compound documents, we have created general and homogeneous conceptual and functional models. In particular, DPI services ensure independence between documents and instruments.

4.3 Software Architecture Models

The model, view and controller components of the MVC design pattern (Krasner & Pope, 1988) are similar to DPI's document, presentation and instrument components. However, MVC, like other architecture models, describe the synchronisation between its components and not a conceptual model of interaction. In contrast, the design of DPI is explicitly driven by the user ontology. The conceptual model is based on the observation that physical instruments mediate interaction between users and documents. The functional model obtains from this instrument metaphor.

5 Conclusion and Future Work

The Documents–Presentation–Instruments (DPI) conceptual model attempts to break the complexity barrier of desktop environments. DPI is based on document-centered interaction, using both a user ontology *and* a designer ontology. The user ontology employs document and instrument metaphors and introduces the *multiple presentations* abstraction. The designer ontology employs properties for persistence, services for user actions and representations for perception, and the concept of compatibility.

This article uses examples to illustrate the expressiveness of the DPI model, and compares it with other systems. However, the DPI model is abstract and must be validated through a concrete implementation. A partial validation has been carried out with the design and implementation of CPN2000 (Beaudouin-Lafon & Mackay, 2000; Beaudouin-Lafon & Lassen, 2000). This application's conceptual model is a simplified version of DPI that addresses a single document type. CPN2000 has implemented and validated independence between documents and instruments, generic instruments and instrument representations.

The next stage is to define a concrete interface model and test it with realistic use cases. After this validation, we plan to implement the full DPI model and test it by building documents and instruments in a real-world setting. We also plan to extend the model to support *groupware* (shared document editing) and investigate other environments such as mobile systems and augmented reality. In summary, the DPI model offers a first but essential stage in designing and implementing a new generation of document-centered environments based on a new interaction paradigm.

Acknowledgement

We thank Wendy Mackay and the anonymous reviewers for comments on earlier drafts of this paper.

References

Apple (1994), OpenDoc Technical Summary, Technical documentation, Apple Computer.

Backlund, B. E. (1997), "OOE: A Compound Document Framework", *ACM SIGCHI Bulletin* **29**(1), 68–75.

Beaudouin-Lafon, M. (1997), Interaction Instrumentale : de la Manipulation Directe à la Réalité Augmentée, *in Actes des 9èmes Journées IHM, Poitiers*, Cépaduès Editions ****SHOULD THE PUBLISHER BE Cépaduès OR Cépadués BOTH OF THESE HAVE BEEN USED BY FRENCH AUTHORS???***.

Beaudouin-Lafon, M. (2000), Instrumental Interaction: An Interaction Model for Designing Post-WIMP Interfaces, *in* T. Turner, G. Szwillus, M. Czerwinski & F. Paternò (eds.), *Proceedings of CHI'2000: Human Factors in Computing Systems*, ACM Press, pp.446–53.

Beaudouin-Lafon, M. & Lassen, H. M. (2000), The Architecture and Implementation of CPN2000, A Post-WIMP Graphical Application, *in* ****EDITOR*** (ed.), *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology, UIST2000*, ACM Press, pp.181–90.

Beaudouin-Lafon, M. & Mackay, W. (2000), Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces, *in* ***EDITOR*** (ed.), *Proceedings of the Conference on Advanced Visual Interface (AVI2000)*, ACM Press, pp.102–9.

Beaudouin-Lafon, M., Berteaud, Y. & Chatty, S. (1990), Creating Direct Manipulation Applications with XTV, *in Proceedings of the European X Window System Conference (EX'90)*.

Beaudoux, O. (2000), Paradigmes et Éléments Architecturaux d'une Boîte à Outils Post-WIMP, Rapport Technique, Computer Science Research Laboratory, École Supérieure d'Électronique de l'Ouest and LRI, Université Paris-Sud (Orsay).

Bier, E. A., Stone, M. C., Pier, K., Buxton, W. & DeRose, T. D. (1993), Toolglass and Magic Lenses: The See-through Interface, *in* ***EDITOR*** (ed.), *Proceedings of SIGGRAPH'93 20th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, pp.73–80.

Brockschmidt, K. (1995), *Inside OLE*, second edition, Microsoft Press.

Buchner, J. (2000), "HotDoc: A Framework for Compound Documents", *ACM Computing Surveys* **32**(1), 33–8.

Celentano, A., Pozzi, S. & Toppeta, D. (1992), A Multiple Presentation Document Management System, *in* ***EDITOR*** (ed.), *Proceedings of the 10th Annual International Conference on Systems Documentation (SIGDOC'92)*, ACM Press, pp.63–71.

Dourish, P., Edwards, W. K., LaMarca, A. & Salisbury, M. (1999), Using Properties for Uniform Interaction in the Presto Document System, *in* ****EDITOR*** (ed.), *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology, UIST'99*, ACM Press, pp.55–64.

Dykstra-Erickson, E. & Curbow, D. (1997), The Role of User Studies in the Design of OpenDoc, *in* G. C. van der Veer, A. Henderson & S. Coles (eds.), *Proceedings of the Symposium on Designing Interactive Systems: Processes, Practices, Methods and Techniques (DIS'97)*, ACM Press, pp.111–20.

Gibson, J. J. (1977), The Theory of Affordances, *in* R. Shaw & J. Bransford (eds.), *Perceiving, Acting and Knowing*, Lawrence Erlbaum Associates.

Gibson, J. J. (1979), *The Ecological Approach to Visual Perception*, Houghton Mifflin.

Gray, D. (1997), *The PhotoShop Plug-ins Book: Category Listings, Instructions and Examples*, Ventana Communications Group, Incorporated.

Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C., Beard, M. & Mackey, K. (1989), "The Xerox "Star": A Retrospective", *IEEE Computer* **22**(9), 11–29.

Krasner, G. E. & Pope, S. T. (1988), "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", *Journal of Object Oriented Programming* **1**(3), 26–49.

Maulsby, D. L., Witten, I. H. & Kittlitz, K. A. (1989), Metamouse: Specifying Graphical Procedures by Example, *in* ***EDITOR*** (ed.), *Proceedings of SIGGRAPH'89*, *Computer Graphics* **23**(3), ACM Press, pp.127–36.

Myers, B. A. (1989), Encapsulating Interactive Behaviors, *in* K. Bice & C. H. Lewis (eds.), *Proceedings of CHI'89: Human Factors in Computing Systems*, ACM Press, pp.319–24.

Norman, D. A. & Draper, S. W. (eds.) (1986), *User Centered System Design: New Perspectives on Human–Computer Interaction*, Lawrence Erlbaum Associates.

Perlin, K. & Fox, D. (1993), Pad: An Alternative Approach to the Computer Interface, *in* ***EDITOR*** (ed.), *Proceedings of SIGGRAPH'93 20th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, pp.57–64.

Shneiderman, B. (1983), "Direct Manipulation: A Step beyond Programming Languages", *IEEE Computer* **16**(8), 57–69.

Sleeter, M. E. (1996), Building Online Help for a Component-Oriented Architecture, *in Proceedings of the 14th Annual International Conference on Marshaling new Technological Forces: Building a Corporate, Academic, and User-oriented Triangle*, ACM Press, pp.87–94.

W3C (1999), XML Path Language (XPath) Version 1.0, W3C Recommendation, W3C.

W3C (2000a), Extensible Markup Language (XML) Version 1.0, W3C Recommendation, W3C.

W3C (2000b), Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation, W3C.

W3C (2000c), XML Query Data Model, W3C Working Draft, W3C.

Author Index

Keyword Index