Majed AbuSafiya, Subhasish Mazumdar

A Document-Based Approach to Monitor Business Process Instances

Keeping track of business process instances is needed for better management, especially queriability and monitorability, of the enterprise as a whole. The business process instances' unpredictable behaviour makes this tracking even more necessary. Currently, this information is not completely or explicitly maintained. We propose a model that captures the states of business process instances by keeping track of their informational access operations (within/outside the scope of automated management). This model is based on an information model that views information within the enterprise as a set of documents and keeps an up-to-date capture of this information model. These models can then be the underlying models for an automated system that keeps track of the states of business process instances and makes this information efficiently queriable.

1 Introduction

Business Process Instances (BPIs) represent the dynamic behaviour of the enterprise. Keeping track of BPIs is required to monitor and control the behaviour of the enterprise as a whole. To provide automated management of business processes [AaHo+03], special information technologies were developed. Two main categories are: workflow management systems ([WFM07], [GeSu+04]) and Enterprise resource planning (ERP) ([Dave98], [Glas98]). Workflow management systems provide automated management of workflows (i.e., automatable parts of business processes). ERP systems are massive software packages that are customized to achieve information integration across the enterprise's functional units. ERP systems provide many benefits like providing a reference model and process templates that embody the current best business practices. ERP systems enhance integration, availability, sharing of information that highly improves the enterprise monitorability.

In spite of the benefits provided by these technologies, BPIs are not completely under the management of these systems. The main reason is that these technologies manage *automatable parts* of business processes that can be precisely defined and whose instances comply with these definitions. However, in reality not all business processes satisfy these conditions. In reality we find that BPIs (1) may include activities that are manual and thus stay outside the scope of automated management. They may also need to access information outside the scope of automated management like documents (e.g., receive student admission application), (2) the default behaviour of the BPI changes due to the continuous change in business process definition. This happens because of the external and internal changes and the need to pursue strategic goals of the enterprise, while business process definitions within the business process management systems are not easy to update accordingly, (3) some BPIs belong to business processes that are vague and hard to define and hence cannot be managed by these automated management systems. Consider for example a student that was suspended, he appealed and a number of meetings and correspondences took place to decide to readmit him. Another example is managerial business processes. In such business processes, there is a sequence of correspondences and meetings, while no precise structuring or definition of the business process is pre-known, (4) BPIs frequently deviate from their default behaviour to deal with special unexpected scenarios, to deal with human errors, temporal changes on the organizational structure, and special cases. Consider for example (refer to as Example 1), the admission office coordinator who receives an admission application for *Alex* who does not satisfy the minimum grade point average (GPA) requirement. However, the coordinator finds, along with the application, a document showing that Alex won the science fair last year. He considers this as a special case and sends a memo to the department and the department replies by requesting to admit Alex. Because of these features, such BPIs need to be completely or partially managed outside the scope of automated

management. Consequently, subsequent querying of the automated system will not reveal the details of the communication and intervention that led to this admission decision. In addition to this limitation, these technologies are expensive to purchase, install, deploy and maintain.

There is a tight association between business processes and information access operations. Many activities are assumed be done only if there is some information within the enterprise information base that proves that. For example, a student is registered only if there is a registration record in the registration database, a student is assumed to be informed about his admission decision, only if there is an e-mail or a letter that was sent to him, the admission office considers a special case admission for a student only if a memo is received from the department. On the other side, information access operations usually occur as a result of some BPI activity (e.g., receiving an admission application in paper form, sending an e-mail, and updating a database record). It is unlikely or may be illegal to access or change the information base of the enterprise without being a part of some BPI.

A second observation is that documents (paper or electronic) compose that important part of the information base of the enterprise that is tightly associated with the ill-behaving BPIs (i.e., those that cannot be managed by current automated management solutions). Documents are that component of the enterprise information base existing outside the scope of automated management systems (databases). Documents are information containers that can easily be created and communicated with any desired informational content for any emerging information or business process management need (consider Example 1). In addition to being the means to manage deviating unpredictable behaviour of business processes, documents may become part of the default behaviour of the business process in case the business process definition itself changes and this change cannot be easily applied in the business process management solutions adopted. Consider for example the policy where a teaching assistant has to pay his/her tuition fees in full before registering. A new change in policy allowed the student to pay through instalments over his stipend period. This change in policy can be captured by an authorization from the student that can be managed through a new form (document) signed by the student without the need to change the business process management system. Moreover, some activities can only be done through documents like communication within and outside the enterprise. Also documents play an important role in vaguely defined business processes since they are our means of coordination, communication and documentation. To summarize, documents play an important role in managing those parts of business process instances that cannot be managed by current business process automated management solutions.

Based on the above observations, we propose to keep track of BPIs by monitoring their effect on documents. Although this may be less useful for automatable parts of BPIs that exist within the scope of automated management, this could be very useful in keeping track of business process instances that are completely or partially outside the scope of automated management. Since (1) documents are the information component of the enterprise existing outside the scope of automated management and (2) documents are tightly associated with deviating and vaguely defined business processes that exist outside the scope of business process management solutions, we can keep track of those parts of BPIs existing outside the scope of automated management by keeping track of their effect on documents. This is also useful for well-behaving BPIs that do not leave a trail within information and business process management solutions. Keeping track of information within the scope of automated management accessed by BPIs is also useful to keep track of BIPs because a BPI may access and span information in the two spaces: information within and outside the scope of automated management (i.e., documents). Retrieval of the latter and the consequent enhancement in monitorability are added benefits.

Although we can keep track of a BPI through logging its activities, there are many advantages in monitoring BPI through keeping track of documents: (1) documents are tightly associated with those parts of business processes existing outside the scope of automated management. The attributes of the involved documents (especially the informational content) are very important to have a more detailed capture of (and hence to reason about) the behaviour of BPIs and hence better queriability and the monitorability of BPIs. However, by logging the activities carried out in the BPI without keeping track of documents, we only keep track of *what* happened with less detail to reason *why* a BPI showed certain behaviour, (2) there is a potential of automation. If we can capture document operations and bind them to the right BPI in a (partially or completely) automated manner the overhead of logging business process activities can be reduced, (3) by considering documents, the logging can be simplified by following the rule: any activity that does not access documents is logged.

To keep track of information access operations, we propose an information model that is based on viewing the information base of the enterprise as a set of documents. BPI's deal with information either in the form of documents (paper or electronic) or in the form of views over information within automated management (databases) that can also be viewed as documents (e.g., the student's registration record

accessed through the screen as a view over the registration database). Based on this document view, we can view the information base of the enterprise as a set of documents and BPIs effect as a sequence of document operations. In the rest of the paper, when we refer to a document we will be referring to this extended definition, that is, views over databases are also assumed to be documents and operations applied on them to be document operations.

The proposed information model is a set of documents that represent an up-to-date explicit capture of the states of corresponding documents within the enterprise. The reality model ***D**** correspond to the information base of the enterprise viewed as a set of documents. So a *d** in ***D****, could be a student's admission application in paper form, an e-mail sent to the department, or the database view of the student admission record. We propose the capture model ***D*** that *ideally* represents an up-to-date synchronous capture of ***D****. That is, for a *d** in ***D****, there is a corresponding document *document(d*)* such that *document(d*)* maintains explicitly an up-to-date information about the state of *d** (Figure 1) . For ***D*** to be an up-to-date capture of ***D****, if a document operation is applied to a document *d** a corresponding document operation should be applied on *document(d*)*. For feasibility reasons, this strict synchronization between ***D**** and ***D*** may be relaxed.

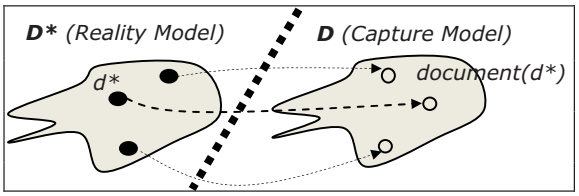

*Figure 1: The Capture of **D*** through **D***

Similarly we propose the capture model ***P*** for the reality model ***P**** (Figure 2). ***P**** is the set of actual BPI's. The execution of a BPI *p** in ***P**** results in a sequence of document operations on documents within ***D****. For every *p** in ***P****, we maintain a corresponding document *process(p*)* in ***P***. The document *process(p*)* maintains up-to-date information about the sequence of document operations that happened on ***D*** corresponding to document operations that happened on ***D**** as part of *p**. This is valid because ***D*** is a synchronous capture of ***D**** and for an operation applied to a document *d** in ***D****, there is a corresponding operation that is applied to *document(d*)* in ***D***. So ideally, ***P*** will maintain up-to-date information about ***P**** by keeping track of document operations that happened as part of BPIs in ***P****.

The whole picture can be seen in Figure 3. We can see a sequence of ***D**** document operations that happened in reality as part of some BPI *p** (the arrows

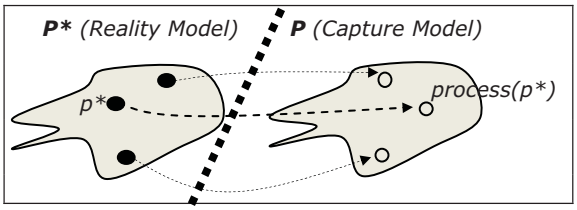

*Figure 2: The Capture of **P**** through ***P***

show the sequencing of these document operations). The capture model ***D*** maintains corresponding *document(d*)* for every *d** accessed by *p** (the vertical arrows). Also applying a document operation on *d** will result in a corresponding operation on *document(d*)* such that *document(d*)* stays an up-to-date capture of *d**. *process(p*)* is a corresponding document that keeps track of ***D*** documents that were accessed and operations applied on these documents as a result of operations applied on ***D**** due to *p**.

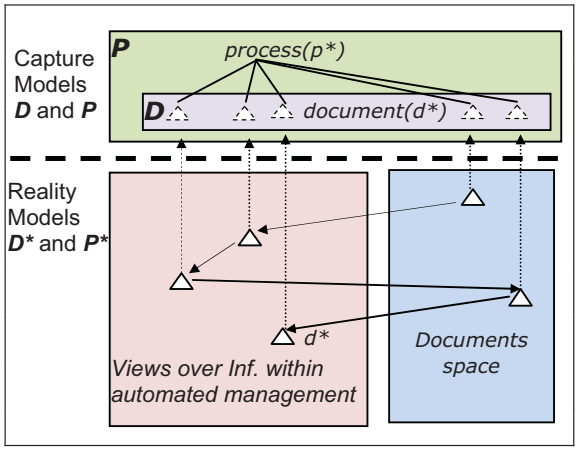

*Figure 3: The Capture of **P**** through ***P***

D and ***P*** can then be the underlying model for a software system that provides automation needed to (1) maintain the capture models up to date by capturing document operations that happen on ***D**** due to ***P**** and update the ***D*** and ***P*** accordingly and (2) provide automated queriability of these models and hence make this information about the BPI states efficiently available whenever needed. So, instead of querying ***D**** and ***P****, we can easier query about them through ***D*** and ***P***.

The rest of this paper is organized as follows: in section 2 we will present the information models ***D**** and ***D***. In section 3, we will present a document-based process definition model and the BPIs model ***P***. In section 4, we discuss the implementability of the proposed system followed by the advantages. We end the paper with concluding remarks and future work.

2 Information Model

2.1 The Reality Model *D**

A document can be defined as a presentable informational container that has certain attributes on which we can apply a certain set of operations. Any information about the document represents an attribute of the document (e.g., informational content, last access, physical location, type). Also, associations with other documents (e.g., version-of) or with other objects within the enterprise (e.g., owner-of which is a participant) are attributes of a document. Documents are objects on which we can apply simple or complex document operation (update, annotate, route, take admission decision). We will use object-oriented language notation to refer to a document attribute or operation, for example, *d.physicalLocation* represents the physical location of the document *d* and *d.print()* refers to print operation.

D* is a set of documents corresponding to the actual information base of the enterprise within/outside the scope of automated management viewed as a set of documents. We have adopted this view because (1) this is the natural view of information where a human and BPI interact with information in the form of documents, (2) it is a homogenous view through which we can view all information within/outside the scope of automated management as documents, and (3) it can be easily transformed into an electronic semi-structured constructs [AbBS00] that are supported by automated management systems ([JaAl+02], [Meye02]).

D* is the target of the BPIs effect. That is, operations that are applied to ***D**** documents are due to some BPI. ***D**** documents can be viewed to exist in three main classes: (1) *paper documents*: Paper documents are still there because electronic documents are frequently printed for annotation, routing, personal use, and signature. Sometimes, only the original paper document is considered to be authentic. Paper documents are also important for communication within/outside the enterprise, (2) *synthesized documents*: There are views over information within automated management, for example, the student registration record from a database presented to the registrar on his computer screen. Considering views over databases to be documents is justified because these views can also be seen as information containers having attributes and on which we apply operations and hence complying with our definition of a document, (3) s*tandalone electronic documents*: These are electronic documents that are not synthesized documents (e.g., word processing files, e-mails, audio or video files). Standalone documents are part of the document space. Note that although electronic documents (whether synthesized or standalone) are presentable on a computer screen, the corresponding *d** is not the screen itself, but the electronic file or the database view shown on the screen.

We restrict ***D**** documents to those documents that are accessed by some BPI. This is justified, because a document is of value only if it is involved in some BPI. In the rest of this paper, when we refer to a document, we refer to any document from the three types specified above except when explicitly specified otherwise.

D* documents can be classified into classes where documents having similar informational content structure and hence are the target of the same set of business processes are assumed to belong to the same *document type*. A document type is a template that defines a general structure of documents with similar informational content structure and behavior. For example, students' admission applications are assumed to be of the same type since they have the same informational content structure and they are the target of a given set of business processes (e.g., admission application processing business process). The types of documents in ***D**** are not explicitly defined, but it is very easy for a human to classify documents into types. We will assume the type of a document *d** to be an attribute of *d** and we will refer to that type as *type(d*)*.

2.2 The Capture Model *D*

Our approach in capturing BPI's in ***P**** is based on keeping track of operations that happen on ***D**** documents. We will define ***D*** to be an explicit up-to-date capture of ***D****. For a *d** in ***D****, *document(d*)* in ***D*** is the corresponding document whose informational content is an explicit representation of the attribute values of *d**. To simplify presentation, if *t* is an attribute of *d** that is captured by *document(d*),* then we will call *t* to be an attribute of *document(d*)*. The explicit representation of the state of *d** by the document *document(d*)* makes this information about *d** easier to map and hence maintained and queried by automated management.

We choose to model *document(d*)* as a semi-structured document that can be represented using XML [AbBS00]. This choice is justified because the semi-structured data provide a flexible information model yet supported by automated management technologies. An attribute value can be viewed as a composite XML element or even a simple element composed of a piece of text or a reference to another element. For example, let *d** be a memo received by the admission office from the department that decided to admit Alex. Assume that the attributes of *d** that we need to keep track of are the informational content and

physical location. We can capture these attributes of *d** by *document(d*)* as shown in Figure 4. Note that (1) *document(d*)* is a document that is an explicit representation of the attribute values of *d**. That is, *document(d*)* is *not* a duplication of *d** (2) *document(d*)* does not necessarily capture all the attributes of *d** (section 2.3.1), (3) how to represent an attribute value depends on the attribute domain chosen. For example, we chose to represent the physical location through department and not building, (4) the attribute values in the example are very simple, but they can be more complex, (5) we will call attributes of *d** that are captured by *documents(d*)* to be an attributes of document(d*). This will make it easier to present *document(d*)* as an image for *d**.

From: CS Dept **To**: Admission Office **Subject:** Alex's Application The department decided to admit Alex. Please be informed. Thanks,	*<memo>* *<informationalContent>* *<from>* CS Department *</from>* *<to>* Admission Office *</to>* *<subject>* Alex's application *</subject>* *<body>* admit Alex *</body>* *</informationalContent>* *<physicalLocation>* Admission Office *</physicalLocation>* *</memo>*

Figure 4: d and document(d*)*

Same document operations applicable to *d** are also operations applicable to *document(d*)*. If an operation is applicable to *d** due to some business process, we assume a corresponding document operation is applicable to *document(d*)* such that it changes the attributes of *document(d*)* exactly as the corresponding operation changes the corresponding attributes of *d**. This is needed to keep *document(d*)* as an up-to-date capture of *d** (section 2.5)

2.3 Defining Types of *D documents*

We need to define ***D*** document types to ease and to allow the automated construction of ***D*** documents and to define business processes. In Section 2.1, we presented the notion of the ***D**** document type. We define a ***D*** document type T_D for a corresponding ***D**** document type T_{D^*} by the following methodology: (1) identify ***D**** documents that belong to T_{D^*}, (2) identify important attributes that we need to keep track of for documents of T_{D^*}, and define corresponding attributes for T_D, (3) identify those document operations applicable to documents of T_{D^*} and define a corresponding operations for T_D. In the following subsections, we will discuss these steps in detail. Let *d** be a document of *type(d*)*, then we can define a corresponding type for corresponding ***D*** documents (refer to as *type(document(d*)*) as follows:

2.3.1 Define the Attributes

Before we show how to define attributes, we need to identify which attributes of *d** that we need to keep track of by *document(d*)*. Not every attribute of *d** needs to be captured, otherwise it would add a lot of complexity in maintaining the model. We only need to consider those attributes of *type(d*)* that are needed to describe the effect of BPIs on the states of documents of this type. That is, business processes that affect a given document type play a very important role in identifying what attributes we need to maintain. For example, let *route* be a document operation that is applied on paper admission applications as part of the admission business process. To capture the effect of this operation we need to define *physicalLocation* as an attribute of *type(document(d*))*. As a general rule, those attributes of *type(d*)* that need to be considered as attributes of *type(document(d*))* are those attributes needed to capture the effect of the BPIs on the state of the document. Said in another way, important attributes are those attributes needed to precisely define the effect of document operations applied on documents of this type as part of some BPI. Documents of *type(d*)* could be involved in more than one business process type. This means that an attribute of *type(d*)* that is not needed to capture the effect of one business process type may need to be defined to capture the effect of another business process type. Newly created business process types may also require defining new attributes or change the definition of an existing attribute, in this case, we can extend the available document types to define new document types (Section 2.3.3).

To define an attribute, we need to define an attribute *name* and define its *type*. We choose to represent attributes as semi-structured data. So an attribute type can be defined by the corresponding hierarchical structure (Figure 5).

For the informational content attribute, we need to define an informational content structure for *type(document(d*))* using which we can map the informational content of *d** into the informational content of *document(d*)*. A mapping from the

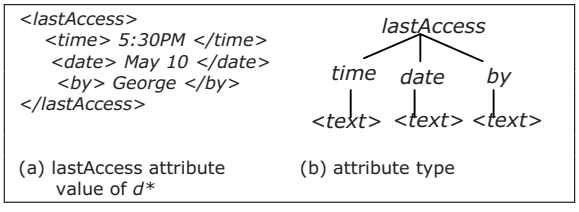

Figure 5: Attribute value and attribute type

informational content of *d** into the corresponding informational content of *document(d*)* needs to be clearly defined. The information objects in *d*.informationalContent* that need to be considered for mapping are those that are required to clearly define the effect of the document operations of the BPIs on *d*.informationalContent*. This is also justified because not all the informational content of *d** needs to be tracked. For example, for documents of *type(d*)* where *d** is an admission application, GPA needs to be considered in defining the informational content attribute of *type(document(d*))* because the definition of *takeAdmissionDecision* operation applied as part of the admission business process depends on the GPA (assume the business rule where a student is admitted if his GPA>3.0). Also *decision* should be considered because it represents the informational content effect of the admission business process on admission application documents. However, the nationality of the student in the admission application may be neglected in defining the informational content attribute of admission application type.

To support document-retrieval-based querying of *d** an electronic copy of *d** can be maintained as an attribute for *document(d*)* (e.g., a scanned admission application, a copy of the electronic copy sent or a database view corresponding to the student admission record). Another important attribute is the *log* that maintains a sequence of log records. When a document operation is applied on *document(d*)* because a corresponding operation happened on *d**, a log record is created and added to the log list. A log record keeps track of information about the document operation occurred (such as operation's name, parameters, time, participant, etc.). This is needed to keep track of BPIs as we will see shortly. Note that many documents especially those used to manage deviating behavior of business processes (e.g., correspondences in Example 1) are not frequently accessed, and hence they don't have large history.

We can also define attributes to capture associations (between documents) that we need to keep track of. Although associations can be defined as separate types (especially, attributed associations), we chose to maintain associations as attributes without the definition of explicit types for them because (1) we allow attributes to be complex information objects, so we still can capture attributed associations (2) we would like to maintain the symmetry between ***D**** and ***D*** at the conceptual level. However, we can define such types when we realize ***D*** by implementation.

2.3.2 Define Operations

Operations to define for *type(document(d*))* can be identified by the following rule: if an operation is applied to documents of *type(d*)* as part of some BPI, define a corresponding operation for *type(document(d*))* such that it affects the attributes of *document(d*)* exactly as the corresponding operation affects the attributes of *d** (note that the attributes here are only the captured attributes). Defining an operation for *type(document(d*))* is defining how the attributes of a document of *type(document(d*))* should change due to this operation.

For example, consider the *takeAdmissionDecision* operation applied on ***D**** documents of type admission application as part of the admission application processing business process. This operation updates the informational content of *d** by annotating the word "accept" if GPA>3.0 or "reject" otherwise. So, we need to define a corresponding operation *takeAdmissionDecision* as an operation for *type(document(d*))*. Note that we need to have *GPA* and *decision* information objects to be defined as part of the informational content of *document(d*)* such that we can define *takeAdmissionDecision* operation. This operation can be defined as follows: (assuming that *getValue* and *setValue* are primitive operations that are already defined):

```
If (getValue(/informationalContent/gpa)>3.0)
    setValue(/informationalConent/decision, "accept")
else
    setValue(/informationalContent/decision,"reject")
```

This shows the effect on the informational content attribute. Any other attributes affected should be updated as well. For example if the *last access* is an attribute of *document(d*)*, then the definition of *takeAdmisisonDecion* should define how this attribute value should be updated.

2.3.3 Extending Other Types

Document type definition is a continuous activity due to the unpredictable behaviour of business processes and the evolving nature of the enterprise. We may also need to refine an existing type to generate a new type. Imagine for example, the need to define a new attribute or operation. We can ease the creation of new ***D*** document types by extending an already defined type (similar to the concept of inheritance in object-orientation paradigm [FoSc99]). The new document type inherits the definitions of attributes and operations of the extended document type. We can then define new attributes and operations or update the definitions of inherited ones.

2.4 Methodology to Build *document(d*)*

A methodology for constructing a *document(d*)* for *d** is

```
1 Bind document(d*) to a matching document type
2 If a matching type is not found
3   Create a new type for document(d*)
4   Bind document(d*) to that type
5 Instantiate document(d*) from d* and its type
```

Binding *document(d*)* to a document type is the first step toward the construction of *document(d*)*. If a matching type is not found, define a new document type and bind *document(d*)* to that type. By binding *document(d*)* to a certain type, *document(d*)* will inherit all the attributes and operations defined for that type. By binding *document(d*)* to a type, we define its structure (i.e., structure of its attributes). Finally, *instantiation* is needed where attribute values are extracted from *d** and transformed into the corresponding attribute values of *document(d*)*.

2.5 Synchronizing *D* with *D**

For ***D*** to be a capture of ***D****, ***D*** should be in synchronization with ***D****, meaning that *document(d*)* should maintain correct and up-to-date information about the state of *d**. Ideally, this can be achieved by maintaining the following rule: if a document operation was applied to *d**, a corresponding document operation should be applied to *document(d*)* such that *document(d*)* remains an up-to-date capture of *d**. In practice, this strict synchronization requirement may need to be relaxed for feasibility and implementation purposes. This relaxation may include not capturing some document operations, or having a delay in application of a document operation on a ***D*** document because of the occurrence of an on a corresponding ***D**** document.

Note that maintaining the temporal order of the captured document operations is very crucial to maintain ***D*** as a valid capture of ***D****. This is mainly because the concurrent execution of (1) individual BPIs (concurrent paths of the same BPI) or (2) multiple BPIs accessing the same *d**. We can maintain the correctness of ***P*** model by maintaining the following rule: the operations applied on ***D***, due to operations captured on ***D****, should be applied in the same temporal order in which corresponding ***D**** document operations happened. It is important to point out that concurrent access of ***D**** documents is safe. For synthesized documents, the database systems will take care of this issue. However, for paper or standalone documents, they are usually accessed exclusively.

*2.5.1 The Correctness of **D** in Capturing **D****

The synchronization between *document(d*)* and *d** might be broken mainly because an operation that happened on *d** was not captured. We can check for inconsistencies between *d** and *document(d*)* by comparing them (maybe manually). Another way to find inconsistencies between *d** and *document(d*)* is to reconstruct *document(d*)* from *d** and compare the new *document(d*)* with the existing *document(d*)*. If there is a difference, it can be fixed by replacing the old *document(d*)* with the new *document(d*)*. That is, reconstruction is a straightforward solution to return to synchronization state. In addition to discovering discrepancies, another advantage of comparing the newly constructed *document(d*)* with already defined *document(d*)* is discovering changes that happened on *d** that were not captured.

In Figure 6, assume a document operation was applied on *d** generating *d*'*. There are two ways to have *document(d*)'* up to date with *d*'*: (1) we know which operation(s) that were applied to *d** so we apply the corresponding operation(s) on *document(d*)*, (2) for some reason we could not capture the operation(s) that happened on *d**. In this case we can return to synchronization by reconstructing *document(d*)* from *d*'* .

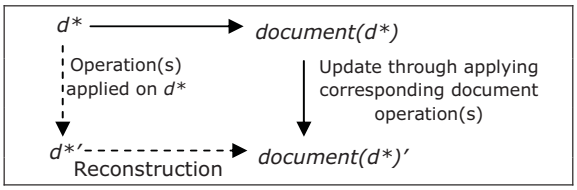

Figure 6: Synchronizing document(d) and d**

3 Business Process Model

3.1 Business Process Definition Model

Based on ***D**** view, we can model a business process by defining a set of document operations and the flow of control among them. This sequence may include branching instructions, repetition or even concurrent behavior. Since ***D*** is the capture model of ***D****, a business process can be modeled by describing the document operations that take place within the business process and the flow of control among them using the corresponding ***D*** document type definitions. We can define a business process as an object-oriented program using typical programming language constructs and whose instructions correspond to document operation calls. In the program we can define variables (or objects) from ***D*** document types (or classes) and define how the states of these variables should change simulating the actual business process effect on these documents. A document operation can be represented as a document operation call applied to a variable bound to a certain document type. One example of an instruction within a program could be *admissionApplication.takeAdmissionDecision()* where *admissionApplication* is a variable bound to *AdmisisonApplication* type. This statement will correspond to take admission decision on a student admission application. Remember that this operation is defined as part of the *AdmssionApplication* document type definition. The program may also use control flow constructs (e.g., *if-then* and *loops*), concurrency structures (e.g., *fork, join*) and data structures *(arrays, lists, etc.)*.

Although uncommon, there could be an activity that is not associated with a document operation (e.g., *make phone call*). We can associate such an activity with a document operation by logging, where the participant carrying out that activity is required to log that he did that activity on some ***D**** document (and hence on a corresponding ***D*** document). In this case the logging operation will correspond to the informational access operation for such activity and hence, our map of activity to a corresponding document access will be complete.

3.2 P

P* is the reality model corresponding to the set of the actual BPIs of the enterprise. ***P*** is a document model that maintains for *p** in ***P**** the document *process(p*)*. In this section we will see how ***P*** maintains an up-to-date model of ***P****.

3.2.1 process(p)*

process(p)* is a document that maintains an up-to-date capture of the state of a BPI *p** by keeping track of (1) every *document(d*)* where *d** was accessed by *p** by maintaining references to every such document and (2) the sequence of document operations applied to these documents as part of *p**. Remember that every document operation that happens to *d** as part of *p** is supposed to be captured and a corresponding operation is applied on the corresponding *document(d*)* resulting in adding a log record to the log attribute of *document(d*)*. *process(p*)* maintains a sequence of references to those log records that were added to ***D*** documents as a result of document operations that happen on ***D**** as part of *p**. *process(p*)* can be realized through an XML document.

Referring to Example 1, the admission coordinator considered Alex's application as a special case and decided to contact the department by sending the document ($d_{request}$) and the department replied with (d_{reply}) to admit Alex. *p** can be captured by *process(p*)*. For a non-deviating BPI, *document*($d_{application}$) and *document*($d_{transcript}$) are first constructed and *process(p*)* is updated to maintain references to these two documents and references to the *receive* log records that were created as a result of applying *receive* operation. For Alex's case, we have another document proving that he won the science fair ($d_{scienceFair}$). Figure 7 shows *process(p*)* initially where it maintains references to *document*($d_{application}$), *document*($d_{transcript}$) and *document*($d_{scienceFair}$). *process(p*)* also maintains a sequence of references to the *receive* operation log record of each of these documents (Figure 8). The deviating behavior of this business process is managed by sending the $d_{request}$ requesting input about this special case. *process(p*)* is updated by constructing *document*($d_{request}$) and maintaining references to this document. When a reply (d_{reply}) from the department is received, *process(p*)* will maintain a reference to *document*(d_{reply}).

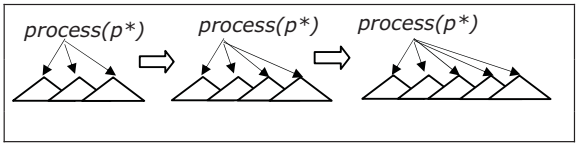

Figure 7: Process(p) maintains references to **D** documents corresponding to **D*** documents*

Note that by capturing a BPI *p** by a document *process(p*)* composed of documents in ***D***, we are actually defining ***P*** on the top of ***D***. Also capturing *p** by the document *process(p*)* will provide a flexible structure for the capture of *p** even if it deviates or belongs to a vaguely defined business process type. Being a document model, ***P*** can be managed using automated management technologies to make information about the states of business process instances queriable.

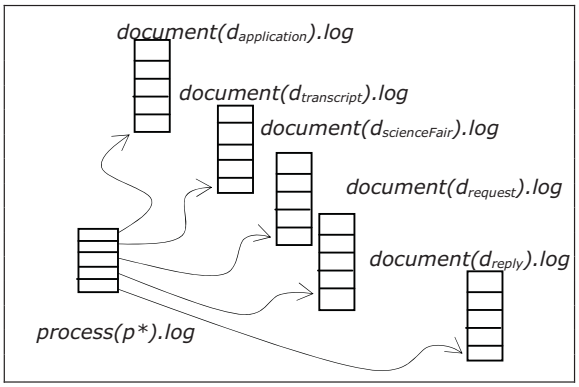

Figure 8: Process(p) maintains a sequence of references to corresponding log records*

3.3 Synchronization between *P** and *P*

For ***P*** to be a valid capture of ***P****, ***P*** should be in synchronization with ***P**** (i.e., *process(p*)* should maintain an up-to-date capture of the state of *p** in ***P****). Part of the synchronization between ***P**** and ***P*** is the synchronization between ***D**** and ***D*** (section 2.5). In addition, *process(p*)* needs to be updated when a document operation on *d** document in ***D**** is captured as part of *p** as follows: (1) maintaining a reference to *document(d*)* if not already maintained by *process(p*)*, (2) updating the sequence of references to log records by adding a new reference to the log record in *document(d*)* that corresponds to the captured document operation.

4 Implementation Discussion

A natural implementation for ***D*** and ***P*** models is using XML and XML databases ([JaAl+02], [Meye02]). ***D*** and ***P*** will be the underlying models of an automated management system whose main requirements is to provide the *automation* needed to (1) capture document operations happening to documents in ***D****,(2) update the underlying models ***D*** and ***P*** accordingly, and (3) provide automated querying capabilities for (***D*** and ***P***). If any of the above requirements cannot be done completely in an automated manner, the system should provide needed user interfaces to get information needed from participants. Note that the proposed system does not replace existing automated management systems (e.g., DBMSs and WFMSs), instead, it will use them and integrate with them (Figure 9). For example, we can use databases to capture operations on synthesized documents and we can utilize WFMSs to keep track of automatable parts of BPIs. We can use these systems to maintain the integrity of ***D*** and ***P*** (e.g., a student with an admission record in the department should have a corresponding BPI in ***P***). Interfacing is needed to provide the automation necessary to capture ***D**** document operations. The *definition component* should provide the interfaces and automation needed to define document and business process types. The synchronization component is responsible for keeping ***D*** and ***P*** up to date with ***D**** and ***P****. It is supposed to process ***D**** document operations capture information to make corresponding updates on ***D***, and bind this document operation to the corresponding *process(p*)*. This component needs to be interfaced with the enterprise information base to capture information access operations within or outside the scope of automated management. Software and hardware resources are needed to automate the capture of document operations and to reduce human overhead.

For paper documents, to keep track of the location of the paper documents, automatically readable (or

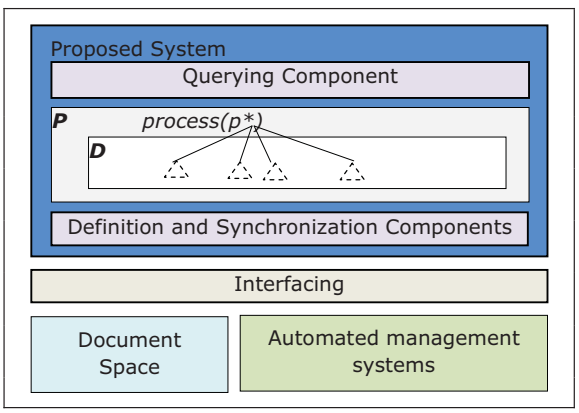

Figure 9: The proposed system

even sensible) identifying tags can be attached and read whenever the document is accessed. Sensors can be installed in important locations within the enterprise to keep track of the document's physical location [AbMa04] (desks and cabinets for example). Cabinets can be supported with locks that open when a participant with a badge is close to keep track of who is accessing the document. By reading this tag of a paper document *d*, document(d*)* can be automatically retrieved and user interfaces can be generated (based on document type definition) to allow the participant to apply the required document operation manually if cannot be captured in an automated manner. However, if *document(d*)* is not already created, user interfaces (based on the document type) can be automatically generated to create *document(d*)*. Participants dealing with paper documents can be supported with scanners to maintain images for paper documents so that a copy of the *d** can be maintained through *document(d*)*.

For *synthesized documents*, the automated management services provided by the database systems can be utilized to keep track of operations applied on them. For example, triggers can be developed to capture operations on database views (e.g., detect the creation of a new record or a change on an existing record).

For *standalone* electronic documents, an integrated standalone document development environment can be developed through which different standalone document development applications can be used (e.g., word processor or e-mail manager). This environment provides available document types (defined by the definition component) where the participant can choose to build a standalone document from an already known type with informational content structured. It should also integrate with the definition component to allow the definition of new types whenever needed. Constructing a standalone document belonging to a specific type allows automatic generation of user interfaces that can be easily instantiated and hence having a faster and more structured construction of a standalone document's informational content. Generating special user interfaces to be filled by a human (when a standalone document is first created or accessed) to associate profile information to the document is already supported by some technologies (e.g., [Word08]). These ideas can be extended to capture more structured detailed information about the document's attributes.

These software and hardware technologies to capture document operations should be interfaced with the synchronization component to update the underlying models. We also need to bind the operation applied on *document(d*)* with the corresponding *process(p*)*. Information about the state of *document(d*)*, the captured document operation, business process defi-

nitions and the states of BPIs in ***P*** could be used to automatically identify the corresponding *process(p*)*. If this binding cannot be done in automated manner, a human intervention may be needed. In this case the system should provide user interfaces needed to get information required to resolve this binding.

The querying component should provide the needed interfaces to create and evaluate queries over ***D*** and ***P*** models. We choose XML database to be the core engine to maintain and query underlying models ***D*** and ***P***. This is justified because of the semi-structured choice to represent the ***D*** and ***P*** models.

5 Advantages of the Proposed Solution

Compared with current automated business process management solutions, the main advantage of the proposed solution is that it keeps track of the states of BPIs even if they deviate from their default behavior, belong to business processes that are vaguely defined, or access or deal with information outside the scope of automated management. Such unpredictable behavior of BPIs cannot be completely managed by the current automated business process management systems. By keeping track of information access operations, and maintaining the underlying models up-to-date, we can query and reason about unpredictable business process behavior. Also because ***D*** maintains information outside the scope of automated management, we can query about information existing outside the scope of automated management by querying ***D*** model. Our approach will complement current IT solutions to provide capture of information and business processes outside the scope of automated management that current IT solutions fails to cover. This would better support *process mining* [AgGu+98] since more information is available than that is provided by automated management solutions.

6 Conclusions

To have better queriability, monitorability and control of the behavior of the enterprise as a whole, we need to keep track of BPIs. Even with the availability of automated business process management solutions, BPIs are not completely under the management of these solutions. An automated solution is needed to maintain a better capture of BPI's especially for those residing completely or partially outside the scope of current automated management solutions. In this paper, we have proposed a model that keeps track of BPIs by keeping track of their information access operations (within and outside the scope of automated management). We view information in the enterprise (information within databases and documents) as a set of documents. We proposed a synchronous capture information model where BPIs information access operations can then be captured by their corresponding effect on this information model. This BPIs model can then be the underlying model for an automated management solution that integrates with current automated management solutions and provides a better capture of BPIs states because the document trail makes it possible to query why something happened in addition to what happened.

The main challenge of the automated solution is capturing document operations. Creative software and hardware solutions are needed to achieve this for paper, standalone and synthesized documents. Binding a document operation to the corresponding BPI is another challenge. Also, smart solutions and techniques are needed to provide efficient queriability of ***D*** and ***P*** models.

References

[AaHo+03] W.M.P, van der Aalst; A.H.M., ter Hofstede; M., Weske: Business Process Management: A Survey. In: Business Process Management, Proceedings of the First International Conference, 2003, pp. 1-12.

[AgGu+98] R. Agrawal, D. Gunopulos, F. Leymann: Mining Process Models from Workflow Logs. In: Sixth International Conference on Extending Database Technology, 1998, pp. 469–483.

[AbBS00] S., Abiteboul; P., Buneman; D., Suciu: Data on the Web: From Relations to Semistructured Data and XML. In: Morgan Kaufmann, 2000.

[AbMa04] M., AbuSafiya; S., Mazumdar: Accommodating paper in document databases. In: Proceedings of the ACM Symposium on Document Engineering, 2004, pp. 155-162.

[Dave98] T., Davenport: Putting the Enterprise into the Enterprise System, Harvard Business Review, 1998, pp. 121-131.

[GeSu+04] U., Gelinas; S., Sutton; J., Fedorowicz: Business Processes and Information Technologies. Thomson Learning, 2004.

[Glas98] R., Glass: Enterprise Resource Planning—Breakthrough and/or Term Problem, The Data Base for Advances in Information Systems, 1998, pp. 14-15.

[JaAl+02] H., Jagadish, S., Al-Khalifa, A., Chapman, L., Lakshmanan, A., Nierman, S., Paparizos, J., Patel, D., Srivastava, N., Wiwatwattana, Y., Wu, C., Yu: TIMBER: A native XML database December 2002 The VLDB Journal, 2002.

[Meye02] W., Meyer: eXist User's Guide. System Documentation Version 0.71, 2002.

[FoSc99] M., Fowler; K., Scott: UML Distilled A Brief Guide to Standard Object Oriented Modeling Language. Addison Wesley, 1999.

[WFM07] Workflow Management Coalition, Workflow reference model. URL: http://www.wfmc.org/standards/docs/tc003v11.pdf, (21 Dec 2007).

[Word08] Worldox, http://www.worldox.com, (5 Nov 2008).

Majed AbuSafiya, Subhasish Mazumdar

Computer Science Department
New Mexico Tech
801 Leroy Place
Socorro, 87801
New Mexico
USA
{majed|mazumdar}@nmt.edu