Bringing Enterprise Business Processes into Information System Products

Naveen Prakash

GCET, 1 Knowledge Park Phase II, Greater NOIDA 201306
praknav@hotmail.com

Abstract. We propose here that enterprise business processes need to be mapped to information systems. Therefore, an information system deliverable is an integration of Enterprise information, Enterprise business rules, and Enterprise processes. Since the process model is built on top of business rules, we propose a uniform representation system that shall explicitly bring out process logic as well as rule logic. As a result, we shall obtain a hierarchy that shall allow smooth movement between process and rule logic. Conceptually, this implies that we shall treat rules as atomic processes. We instantiate the generic method model with concepts of the IS deliverable and thereby represent rule and process logic as dependency graphs. We exemplify our representation with the ATM bank example and show a verification step to ensure that enterprise process needs are indeed met.

Keywords: Enterprise Business Process, Information System etc.

1 Introduction

An enterprise model has been defined [Fox97] as a "computational representation of the structure, activities, processes, information, resources, people, behaviour, goals and constraints of business, government, or other enterprise". The framework of Enterprise Architecture of Zachman [Zac93] provides a structure for classifying and organizing the description of an Enterprise for both, enterprise management and enterprise systems development. Enterprise modelling [Lou95] has been applied in such diverse areas as Computer Integrated Manufacturing, Enterprise Integration, Business Process Re-engineering and Information Systems Engineering. Our interest here is in enterprise modelling from the Information Systems perspective.

Modeling enterprise processes is a well recognized research problem. Keller and Detering [Kel96] have pointed out the essential dilemma faced when using an ERP system, namely, whether to adapt to the software and radically change business practice or to modify the software to suit enterprise needs. Obviously, the latter is ruled out once investments in ERP packages like SAP [ASA99] are made. While highlighting the importance of enterprise processes, Dalal et al [Dal04] pointed out their crucial role in the next generation of ERP systems, christened ERP II.

Loucopoulos and Kavakli [Lou95] establish a relationship between enterprise modeling and requirements engineering and suggest that the enterprise-information

J. Stirna and A. Persson (Eds.): PoEM 2008, LNBIP 15, pp. 168–181, 2008.

systems relationship provides justification criteria and explanations about the information system to be developed. However, Requirements engineering [Myl92] is essentially concerned with functional and non-functional requirements and process model elicitation, has to our knowledge, not been addressed. Thus, it is not possible to deal with enterprise process models in Requirements engineering.

From the foregoing, it can be seen that, in enterprise modelling, we have to represent both, the information/data aspect of enterprises and their process aspects. The completeness principle postulated in [Pra07] provides a basis for this. This principle implies that the deliverables of IS/SW activity are information/data models and associated usage process models. During enactment, the latter produces the former. Notice that the completeness principle gives primacy to the process model over system functionality. Indeed, a process model is built over functionality but the argument is that it is enterprise processes that deliver value and therefore, are paramount.

We refer to the integration of information/data model and process model by the relatively neutral term, information system deliverable or IS deliverable. Essentially, we are dealing with three phenomena that are to be integrated together in the IS deliverable. These are

- Enterprise information,
- Enterprise business rules, and
- Enterprise processes.

A number of techniques, data-oriented, function-oriented, behaviour-oriented and object-oriented, have been developed for representing enterprise information types and business rules. Since the process model is built on top of business rules, we shall look for a uniform representation system that shall explicitly bring out process logic as well as rule logic. As a result, we shall obtain a hierarchy that shall allow smooth movement between process and rule logic. Conceptually, this implies that we shall treat rules as atomic processes.

Once business rules and processes are treated in the same uniform manner, we need to address the question of the level of abstraction of their representation. BPML {Ark02], BPEL [OAS07] and workflow[wfm95] provide features for representing sequence, parallelism and choice in process logic, fault and exception handling, synchronization and other such features. We believe that there is a higher level of abstraction at which enterprise process models should be represented that brings out their essential features and acts as a specification for the low level representation. Thus, synchronization, faults/exceptions are left to be elaborated in subsequent stages.

We choose the generic method model [Pra06] as the basis for instantiation of the IS deliverable. This model has integrated the information and process parts of methods together. For the latter, it produces a dependency graph based on different types of dependency. Thus, the dependency graph captures a range of process situations. We represent the enterprise process model as a dependency graph and shall refer to this representation as the Application Process Model, APM.

The layout of the paper is as follows. In the next section we present the generic method model and highlight the features that we shall use. In section III, we develop a set of concepts to represent the IS deliverable, instantiate the generic model with these and show the use of the dependency graph for representing business rules and process models. In section IV we apply the notions we have developed to model the ATM of a

bank. Section V introduces a verification step called matching for ensuring a good fit between the APM and the enterprise business model. It uses the ATM example to show how matching can be done. In section VI we compare our approach with others like workflow, BPML etc.

2 The Generic Method Model

In this section, we provide an overview of the generic method model so as to show its usefulness in building the IS deliverable. The details of the model can be found in [Pra06]. As its name implies, this model was developed to capture the generic notion of a method. One of its stated objectives is to integrate the product and process aspects of methods together. Thus, if we treat an IS deliverable as a method, identify its concepts and instantiate the generic method model with these, then the IS deliverable shall have integrated product and process parts together with all other properties of the generic model.

The generic view treats a method as a triple <M, D, E> where M is the set of method blocks, D is the set of dependencies between method blocks, and E is the enactment mechanism. The notion of a dependency is used to build a dependency graph with method blocks as nodes and dependency types as edges.

A method block has two parts, an argument part and an action part. The action part acts upon the argument part to produce the product. These two parts correspond to product primitive and process primitive respectively of Fig. 1. The product primitive is found in the product model. Therefore, if the product model is the schema of an application, then the concepts of this schema form the product primitives. The process primitives correspond to the operations allowed on the concepts in the schema. For example, <Reservation, Create> is a method block.

Complex method blocks are built out of simpler ones. For example, <Name, Address, Start, End, Room type, Book> is a complex method block, built out of simpler method blocks that create a client, check availability of rooms and reserve a room respectively. Here, Book is the action part and the rest are arguments.

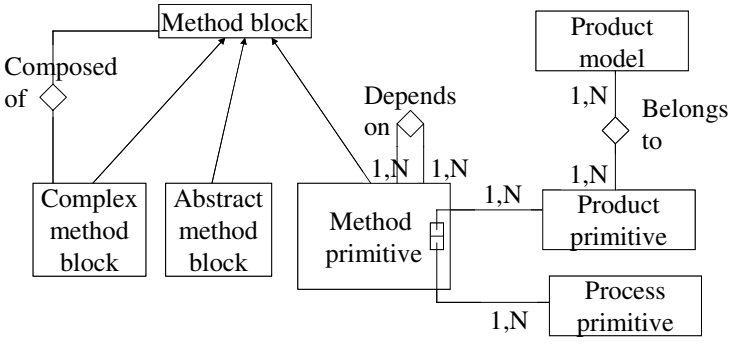


Fig. 1. The Generic Model

Now consider the notion of a dependency. Two attributes, urgency and necessity, are associated with each dependency type. Urgency refers to the time at which the dependent method block, O_2, is to be enacted. If O_2 is to be enacted immediately after O_1 is enacted then this attribute takes on the value *Immediate*. If O_2 can be enacted any time, immediately or at any moment, after O_1 has been enacted, then urgency takes on the value *Deferred*. Necessity refers to whether or not the dependent method block O_2 is necessarily to be enacted after O_1 has been enacted. If it is necessary to enact O_2, then this attribute takes the value *Must* otherwise it has the value *Can*. As discussed earlier, this gives rise to four dependency types displayed in Table 1.

Table 1. Dependency Types

Type	Urgency	Necessity	Abbreviation
1	Immediate	Must	IM
2	Immediate	Can	IC
3	Deferred	Must	DM
4	Deferred	Can	DC

It can be seen that the urgency property of a dependency type supports both instantaneous and long running phenomena. If the dependency between a pair of method blocks has Urgency=Immediate then the two are to be immediately enacted, one following the other. This amounts to a representation of instantaneous phenomena. On the other hand, if Urgency=Deferred then there can be a time interval between the enactment of the two method blocks and one can represent long running phenomena.

The necessity property of a dependency type supports planned, determined action or selection of alternative actions. If the dependency type between O1 and O2 has Necessity=Must, then O2 must be enacted after O1. This corresponds to a planned, determined course of action. On the other hand if Necessity=Can, then O2 represents a choice of action.

Using the notion of a dependency, a method can be organized as a dependency graph. This graph when traversed from its start to stop nodes represents an activity that can be performed by the system and can be seen to be a process model. As a result of process enactment, the system produces an instance of the product model, the product.

We illustrate a dependency graph by considering a method with the set of method primitives $O = \{O_1, O_2, \ldots, O_{14}\}$. Let there be two dependency types (see Table 1), one of type 1 and the other of type 2. Let the following dependencies be defined:

IM dependencies

$O_1 \rightarrow O_2$ $O_1 \rightarrow O_3$ $O_1 \rightarrow O_4$ $O_1 \rightarrow O_5$
$O_6 \rightarrow O_7$ $O_6 \rightarrow O_8$
$O_9 \rightarrow O_{10}$ $O_9 \rightarrow O_{11}$ $O_9 \rightarrow O_{11}$

IC dependencies

$O_1 \rightarrow O_6$ $O_1 \rightarrow O_9$
$O_6 \rightarrow O_{13}$ $O_6 \rightarrow O_{14}$

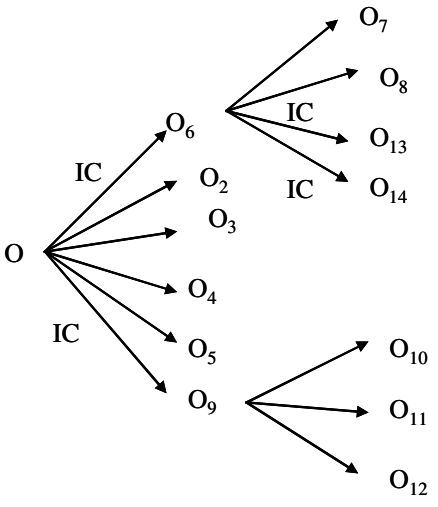


Fig. 2. A Dependency Graph

Using the foregoing dependencies, we arrive at the dependency graph as shown in Fig. 2. In this Figure, only the IC dependencies are labeled and the non-labeled ones are assumed to be IM dependencies.

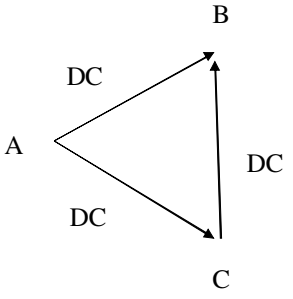
The dependency graph shows that the entire activity represented in it is instantaneous. Further, there is choice to enact O_6 and O_9 after O as well as O_{13} and O_{14} after O_6.

Enactment Initiation and Termination. A dependency graph has a set of nodes, called START, that have no edges entering them. This implies that enactment can begin from any of the nodes in this set. For example, for Fig. 2, START contains exactly one node, O, and enactment begins from this node.

Now consider termination of enactment. We define a set STOP that contains nodes at which enactment can terminate. The following nodes belong to this set:

1. Nodes that have no edges coming out of them. For example, in Fig 2, O_7, O_8 and O_{10} to O_{14} shall be members of STOP.
2. Nodes that have edges leaving them but all these edges have Necessity = Can. Since the edges identify nodes which are optional and may not be enacted, it is possible for enactment to terminate. Notice that even if one of the edges has Necessity=Must then termination cannot occur since the node determined by such an edge is to be necessarily enacted.

For example, consider the dependency graph shown below. There are no nodes leaving B. Therefore it is a member of STOP. A has two edges leaving it and both have Necessity=Can. Therefore, it is possible that these may never be enacted, and enactment may stop at A. Therefore, A is a member of STOP. Similarly, the edge leaving C has Necessity=Can and B may never be enacted after C. Therefore C is also part of STOP.



From the foregoing we get STOP = {A, B, C}. That is, enactment may stop at A, after the enactment of AB, of AC, or of ACB.

Let us determine STOP for the dependency graph of Fig. 2. By rule 1 above, we get O_7, O_8 and O_{10} to O_{14}. Further an examination of the graph shows that only two out of the six edges leaving O have Necessity=Can and the rest Must be enacted. Therefore, O is not a member of STOP. All edges coming out of O_9 have Necessity=Must. Therefore, it is not a member of STOP. Similarly, O_6 is not a member of STOP because only two of its four edges have Necessity=Can. Therefore, we get STOP = {O_7, O_8, O_{10}, O_{11}, O_{12}, O_{13}, O_{14}}.

3 The IS Deliverable

As mentioned earlier, our view of an IS deliverable is based on the completeness principle. This completeness principle [Pra07] states that an information system deliverable should be a faithful representation of the product and process models at the required level of conceptualization. In other words, applications can be visualized at different levels of abstraction and at each level the IS deliverable is complete if and only if both the product and process models have been developed. Our interest here is not in the different levels of abstraction but in the representation of the product and process aspects of an IS deliverable.

Looking at the generic method model from this perspective, we notice that the product primitive comes from the product model. We first need to conceptualise product primitives and associated process primitives of an IS deliverable and thereafter, instantiate the generic model.

In information systems, the data part of a deliverable is modelled as a conceptual schema. The conceptual schema contains application specific types of information that we refer to as Application Product (AP) types. The conceptual schema can be populated with instances of AP types by operations of create, delete and modify. We refer to these operations as Operations on AP or OAP for brevity. For example, Student is an AP type and read, create, delete, and modify are OAPs. AP types and OAPs come together to form the basic 'action' capability that leads to product construction. For example, Create Student populates the product with students. We shall refer to this coupling as an application primitive.

Application engineers build system functionality using application primitives. For example, we may have AP types Student and Enrolled_in and build method blocks Create student and Create Enrolled_in. We can define Admit Student over Create student and

Create Enrolled_in. We refer to operations like Admit that provide functional capability as User Operations, UO.

In the next section, we instantiate the generic method model with the concepts, AP type, AOP and UO found in the product model of the IS deliverable.

3.1 Instantiating the Generic Model

The instantiation of the generic model is shown in Table 2. A product primitive is instantiated with AP type, whereas a process primitive is instantiated with OAP. To instantiate a method block we define the notion of an application chunk. An application chunk represents the operations that can be performed on the AP types of the application. It is the mechanism for creation of the product part of the information system deliverable and can be used as a node in the dependency graph.

We instantiate method block with the notion of application chunk. An application chunk represents the capability to build the product. A method primitive is instantiated with application primitive introduced above. An application primitive represents the smallest meaningful product building action that can be performed in the application.

Table 2. Instantiating the Generic Model

Generic Method Concept	Application Method Concept
Product primitive	AP type
Process primitive	Operation on AP (OAP)
Method Block	Application chunk
Method primitive	Application primitive, <AP type list, OAP>
Complex Method Block	Complex application chunk <AP type list, UO>
Abstract Method Blocks	Abstract UOs

A complex method block is instantiated by a complex application chunk, CAC. The CAC is built over other simpler application chunks. The argument list and action of a CAC is useful to represent the coupling <AP type list, UO>. Thus application functionality can be represented as a CAC.

An abstract method block is instantiated with abstract application chunk. Such a chunk abstracts out the common features of application chunks and allows us to look at chunks at different levels to construct an ISA hierarchy of application chunks.

Using the four kinds of dependencies, we can now build an application dependency graph with application chunks as nodes and edges that represent their ordering. Edges are labeled with their dependency types. There are two interesting graphs

1. The Function Graph, FG: The distinguishing feature of a FG is that its nodes are application primitives only. We can treat FG as a complex application chunk of the form <AP type list, UO> by associating a UO with it and determining the AP types involved.
2. The APM graph, APMG, constructed with a mix of the three different types of application chunks. As its name implies, this graph can be used to represent a usage process model.

These two kinds of graphs form a hierarchy of graphs. At the root, is the APMG that captures the global process model, the APM. Nodes in this graph can be decomposed into FGs, possibly through a multi-level decomposition hierarchy of graphs.

3.2 FG for Representing a Business Rule

We show that a function graph can be used to capture a business rule. To illustrate this, consider the business rule for making a reservation, "Obtain full requestor information before doing the reservation". Let the AP types be Requestor, Reservation, and Availability. Upon getting a booking request, we can instantiate these using application primitives, Create Requestor, Create Reservation, Modify Availability. A dependency graph for this is shown in Fig. 3.

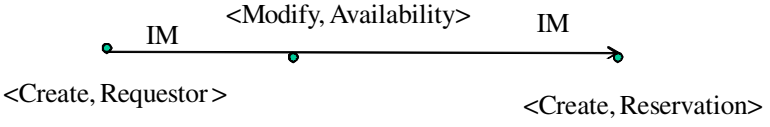


Fig. 3. An FG for a Business Rule

The business rule is instantaneous; the entire execution from the first to the last is done at the same time. Now consider a different business rule that expects the reservation to be carried out before obtaining requestor information. This calls for a different dependency structure among the application primitives as shown in Fig 4.

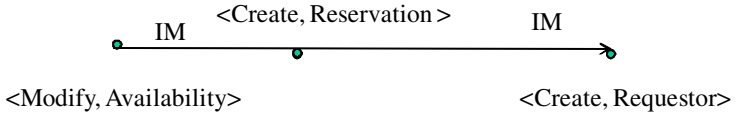


Fig. 4. Representing a Different Business Rule

Again, this is an instantaneous function. Notice that both the dependency graphs can be seen as modelling the business function, Make reservation; both represent instantaneous phenomena, and both will have the same arguments. However, the business rule followed in the two cases is different. Naturally, that dependency structure that best meets the business rules of the enterprise is t be selected. Notice also that Make reservation is a complex application chunk.

As an example of a function graph that captures long-running phenomena consider development of a complex application chunk for admitting a student. Candidates apply for admission and some of these who are prospective students are selected to appear for a test and interview. Based on performance in these, selected students are identified and offered admission. The dependency graph is shown in Fig. 5. Notice that the dependency type between Create Candidate and Create prospective student is Deferred Must. Since the urgency property of this dependency type is deferred, the dependency graph captures a long-running business rule.

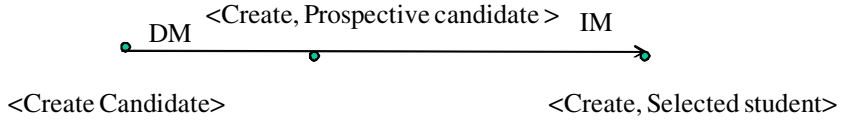


Fig. 5. Representing a Long running Business Rule

3.3 APMG as Process Model

Having looked at the representation of enterprise business rules, let us now consider the enterprise business process model. The application chunks with UOs are the building blocks of such a model. Consider the set of application chunks

ACset = {Ac1, Ac2, …. Acn}

A number of process models can be built each with a different ordering of ACset. Out of these, only some are feasible. Further, out of the feasible operations only some are compatible with the required enterprise business process of the organisation. To illustrate, consider a simple hotel reservation application having

ACset = {Make Reservation, Cancel Reservation, Postpone reservation}

Out of the different orderings possible, the useful one is that which starts with Make, followed by either Cancel or Postpone.

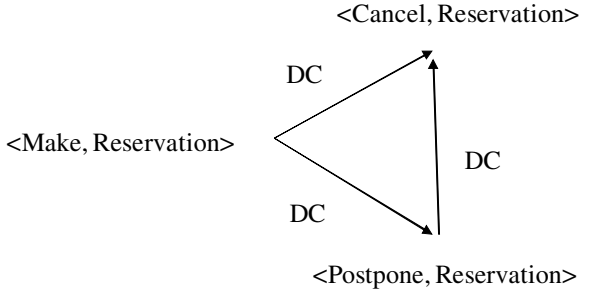


Fig. 6. An APMG for Reservation Process

The dependency graph of Fig. 6 shows that the business process can stop at three nodes, at cancellation, at postponement or at making the reservation. Thus we can have four processes (a) Make reservation, (b) Make reservation, cancel reservation (c) Make reservation, postpone reservation, cancel reservation (d) Make reservation, postpone reservation Further, if it is possible to postpone more than once then the dependency graph needs to be augmented by a self loop from Postpone reservation to itself with the dependency type DC.

The process model represented by the dependency graph is a long running process (Urgency = Deferred). Further, cancellation and postponement of a reservation are choices (Necessity=Can). Notice that as the process model is enacted, the product model of the reservation system is instantiated.

4 The ATM Example

In this section we illustrate our proposals with the help of the example of an ATM of a bank. The information part of the application product model of the ATM is shown in Fig. 7. As shown customers have a name and hold accounts in our bank. Accounts have a number, Acno, and balance. A personal identification number, pin, is associated with each account held by a customer.

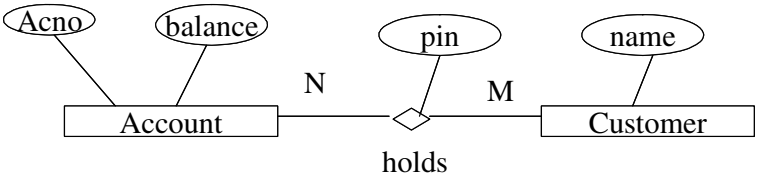


Fig. 7. An IS/SW Product

Now, let us look at the second part, the business rules part, of the IS/SW deliverable. Consider withdrawing money from the ATM. After reading the account number, Acno, and the pin of the customer, withdrawal can be done in either fast or normal mode. In the latter case a statement of the balance and withdrawal details are printed out. The dependency graph is shown in Fig. 8.

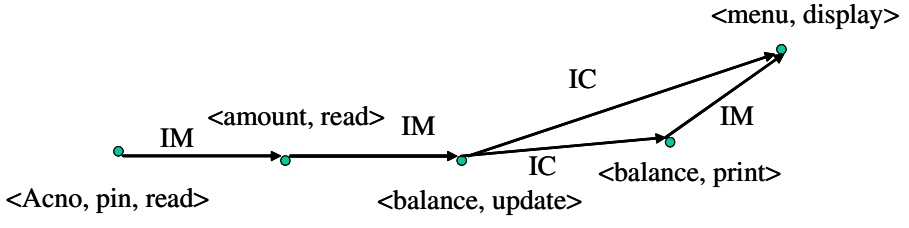


Fig. 8. A Function Graph

The fast and normal alternatives are expressed as IC dependencies. Once the request for withdrawal is satisfied, the main menu of the ATM is displayed again. Fig. 8 can be seen to be a complex application chunk <Acno, pin, amount, withdraw> having AP type list consisting of Acno, pin, and amount whereas the UO is withdraw. It can also be seen as capturing the business rules to withdraw cash from an ATM. In a similar manner, we assume that application chunks to change the pin and check balance respectively have been defined.

We can consider the third part, the process part, now and build a APMG for our ATM example. For this, we assume the set of application chunks consisting of (a) inserting a card into the ATM, <card, insert>, (b) selecting a language, <language, select>, (c) withdrawing amount <Acno, pin, amount, withdraw>, (d) changing the pin <Acno, pin, npin, change>, and (e) checking balance, forming <Acno, pin, check>. These application chunks can be put together in the APMG graph shown in Fig. 9. First notice that all edges have Urgency=Immediate. This means that the

entire APMG is considered instantaneous, to be performed at the same moment. After inserting the ATM card, the language is selected. After that any one of withdraw, check, or change UOs can be performed. One possibility is to exit after performing the chosen UO. The other option is to perform yet another operation from among these. Thus, for example, the sequence of check followed by withdraw could be performed. These alternatives are shown by the IC dependency types associated with edges between application chunks.

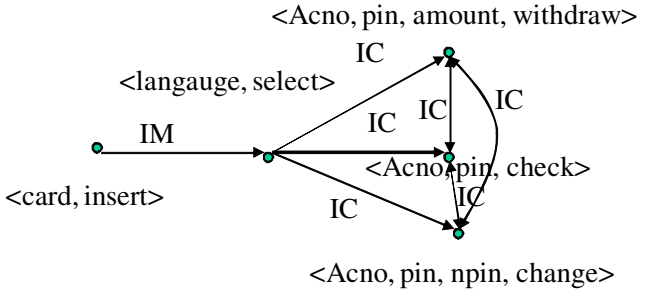


Fig. 9. An Application Process Model Graph

It can be seen that graphs are organized in a hierarchy. In our example, Fig. 9 is the root graph of the hierarchy. One branch of this hierarchy is the graph corresponding to withdraw and the other branches correspond to the graphs for changing pin codes and checking balances (not shown in this paper).

5 Performing Verification by Matching

Once the dependency graphs have been built, we propose that they be verified against enterprise requirements by carrying out a verification activity. Essentially this involves matching enterprise needs against the dependency graphs. We consider this for our ATM example.

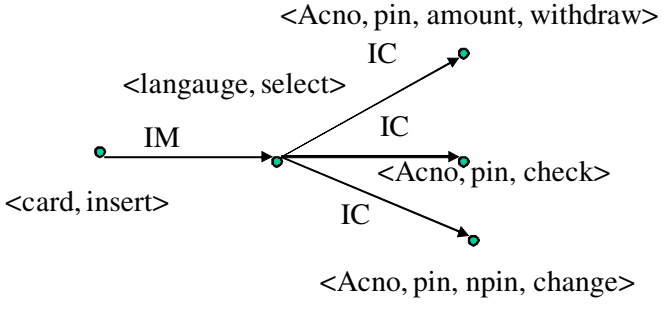


Fig. 10. Selected variants for an Inconvenient ATM Process

There are two strategies that can be followed in enterprises when considering an ATM. One is to offer greater protection to the customer at the expense of convenience. Thus, to protect forgetful customers who may leave their ATM cards in the ATM machine, each operation explicitly requires card insertion. To support such an enterprise process, the APM of Fig. 10 suffices. It requires the removal of three edges in the original APMG of Fig. 9.

The second business strategy could be to assume less forgetful clients and lay greater emphasis on convenience. Then, the APMG of Fig. 9 is accepted as such. Notice that the pin has to be entered for each operation for security purposes but a sequence of UOs can be carried out without having to insert the ATM card many times.

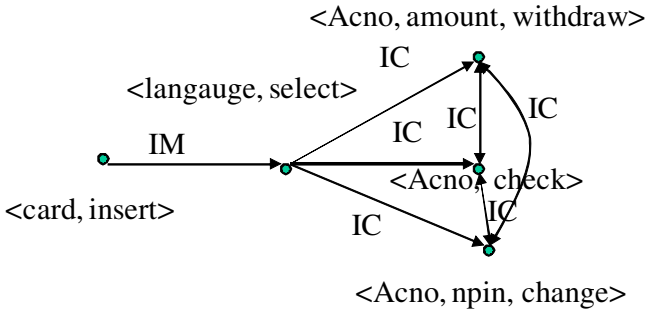


Fig. 11. An Insecure ATM Process

A third strategy could be a variant of the second one. It does away with entering pin each time a UO is to be performed. This leads to an ATM process that is less secure than the one of Fig. 9. Here, it is possible for the client to forget the card in the machine and for somebody else to walk up and use it!! This APMG is represented in Fig. 11 and it requires the modification of nodes. It forces the application engineer to redesign the UOs comprising the APMG to ensure a good fit.

6 Related Work

The workflow approach covers the design, control and execution of business processes. Workflow management systems have been developed to deal with different kinds of workflows, production, administrative, and collaborative. A 3-dimensional framework has been used to bring out the three aspects that go into defining workflows, cases, resources, and tasks. A task-case pair represents a work unit and the logic of this work-unit is the key issue. Workflow logic has been expressed [wfm95] in four basic constructs, sequence, AND-split and join, OR split and join, and iteration. Organization modeling for resource classification and structuring has also been done in the workflow area.

Notice that the 3-dimensional framework does not consider the issue of the output of the workflow, what is the result, the 'information product' produced. Clearly, it deemphasizes the integration of the information product and the process that produces it.

A second approach is business process modeling. BPML [Ark02] has been developed as a means to represent business process models. BPML can be used for expressing abstract and executable processes. Similar to workflows, features for expressing sequences, choice, iteration and parallel execution are available. There is also a full range of facilities for fault/exception handling. Again, the focus is on the representation of business process logic and the result of this logic, the informational products that fall out of these processes are not considered.

A third approach has been developed in the context of web services and their capability to support business processes. WS-BPEL [OAS07] is a language that captures the operational characteristics of business processes. The logic of the process can be implemented and support for execution is provided by an orchestration engine. BPEL code uses constructs similar to those of a general purpose programming language together with full error/exception handling capabilities. BPEL modelling tools have been developed from which BPEL code can be generated. Again notice the lack of emphasis on the information product produced.

Whereas workflow, BPML and BPEL are highly oriented towards enterprise and business process logic, information systems development has de-emphasized it. Early information system products were either data or function oriented, but today a balance between data, function and dynamic aspects of information systems has been found. Consequently, the data aspect of the system being represented is closely coupled with the operational capability that affects the data. Thus, from the point of view of information system developers, the product to be delivered is this combination of data + function. This view ignores the representation of enterprise process models.

It can be seen that whereas information system developers do not pay sufficient attention to enterprise processes, the workflow/business process/web service developers do not pay sufficient attention to modeling the data to be maintained in the system. Our approach can be seen as a way to lay equal emphasis on these twin aspects of enterprises in the design of information system deliverables.

A somewhat different approach is adopted in ERP systems where a cross-module and intra-module process model is assumed. An enterprise using the package must adapt to the prescribed process model. Thus, instead of the system reflecting enterprise process requirements the situation here is the reverse.

7 Conclusion

In this paper we have considered the representation of enterprise business processes in information systems. Information Systems of today provide functionality independent of a process model. This leaves open the possibility of enacting functionality in an undefined order. This problem gets aggravated when the number of functions is high, as it is when doing enterprise modeling, To avoid this, one can build appropriate constraints in the functions but this amounts to ‘hard coding’ process knowledge. Clearly, this hides process information and makes for change resistance.

Seeing the close relationship between business rules and business processes, we represent these in the same uniform notion of a dependency graph. Thus, we get the same properties for both. For example, it is possible for both, business rules and processes, to be long running.

In order to effectively realize the notion of an information system deliverable we see some open spaces and scope for future

- Requirements engineering techniques need to be augmented to elicit process model needs. This is to be done uniformly both for business rules and enterprise processes.
- Just as notions of product customization and adaptation have become acceptable, we need to develop the notions of process customization and adaptation.
- Information system development methods today have the capacity to support construction of application products, data and functionality only. If methods are to produce the information system deliverable, then method engineering tools and techniques shall have to be extended to produce methods capable of building IS/SW deliverables.

References

[Aal98] van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems and Computers 8(1), 21–66

[Ark02] Arkin, A.: Intalio, Business Process Modelling Language (November 2002), http://www.bpml.org

[ASA99] ASAP World Consultancy and Blain, J., et al., Using SAP R/3, Prentice Hall of India (1999)

[OAS07] OASIS Web Services Business Process Execution Language version 2.0, http://oasis-open.org

[Dal04] Dalal, N.K., Kamath, M., Kolarik, W.J., Sivaraman, E.: Towards and Integrated Framework for Modeling Enterprise Processes. CACM 47(3), 83-83-87

[Fox97] Fox, M.S., Gruninger, M.: On Ontologies and Enterprise Modelling. In: International Conferene on Enterprise Integration Modelling Technology, Italy

[Kel96] Keller, G., Detering, S.: Process-oriented Modeling and Analysis of Business processes using the R/3 reference Model. In: Bernus, P., Nemes, L. (eds.) Modeling and Methodologies for Enterprise Integration, pp. 69–87. Chapman & Hall, Boca Raton (1996)

[Lou95] Loucopoulos, P., Kavakli, E.: Enterprise Modelling and the Teleological Approach to Requirements Engineering. Int. J. Cooperative Information Systems 4(1), 45–79

[Myl92] Mylopoulos, J., Chung, L., Nixon, B.: Representing and Using Non-Functional Requirements: A Process-Oriented Approach. IEEETSE 18(6), 483–497 (1992)

[Pra07] Prakash, N.: Complete methods for building complete applications. In: Ralyte, J., Brinkkemper, S., Henederson-Sellers, B. (eds.) IFIP WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences, pp. 207–221. Springer, Heidelberg (2007)

[Pra06] Prakash, N.: On generic method models. Requirements Engineering Journal 11(4), 221–237 (2006)

[wfm95] The workflow Reference Model, http://wfmc.org

[Zac93] Zachman, J.A.: Concepts of the Framework for Enterprise Architecture, Zachman International