A Layered Software Specification Architecture

M. Snoeck, S. Poelmans, and G. Dedene
Management Information Systems Group
Katholieke Universiteit Leuven,
Naamsestraat 69, 3000 Leuven
email: {monique.snoeck, stephan.poelmans,
guido.dedene}@econ.kuleuven.ac.be

Abstract. Separation of concerns is a determining factor of the quality of object-oriented software development. Done well, it can provide substantial benefits such as additive rather than invasive change and improved adaptability, customizability, and reuse. In this paper we propose a software architecture that integrates concepts from business process modeling with concepts of object-oriented systems development. The presented architecture is a layered one: the concepts are arranged in successive layers in such a way that each layer only uses concepts of its own layer or of layers below. The guiding principle in the design of this layered architecture is the separation of concerns. On the one hand workflow aspects are separated from functional support for tasks and on the other hand domain modeling concepts are separated from information system support. The concept of events (workflow events, information system events and business events) is used as bridging concept between the different layers.

1. Introduction

The proponents of object-oriented software development attribute a number of qualities to object-oriented software development such as improved adaptability, maintainability and reuse. In spite of more than a decade of experience with object-oriented technology, the improvements are not really overwhelming. Although separation of concerns is recognized as a determining factor for the adaptability and maintainability of software, there is not yet a generally accepted way to achieve this. Separation of concerns can be pursued at different levels of abstraction in the software development process. In this paper we present a layered software architecture that represents a separation of concerns at a high level of abstraction: it classifies specifications in different layers such that each layer only relies on concepts of the same layer or on concepts of the layers below. At the same time, this architecture outlines a basic implementation architecture. To obtain this layered architecture, we start from the assumption that a full-fledged information systems development method should take all aspects into account: aspects of business process modeling and aspects of the functional part should be linked together. Because most object-oriented analysis and design methods do not yet integrate business process modeling aspects in an adequate way we motivate this assumption and briefly present the approach in section 2. In section 3 we present the four basic layers of our architecture. In section 4 these layers are further refined. Finally section 5 presents some conclusions and topics for further research.

A.H.F. Laender, S.W. Liddle, V.C. Storey (Eds.): ER2000 Conference, LNCS 1920, pp. 454-469, 2000.

2. The Need for Integration of Workflow Aspects in Information System Modeling

The layered architecture proposed in this paper encompasses all the aspects of IT-support in an organization. On the one hand there is the required support for supervising, recording and controlling business activities. On the other hand there is the required functional support for these business activities. The first type of support constitutes the Workflow System, whereas the functional support constitutes the Business Information System. Current object-oriented analysis and design methods focus primarily on the analysis and design of business information systems. In this paragraph we first look at the requirements for adequate business process support. We then argue the statement that current object-oriented analysis and design methods lack a business process view. Finally, we briefly explain how the concept of "business event" can be used to link concepts from business process modeling with object-oriented analysis and design concepts.

2.1 Requirements for an Adequate Workflow System Support

In the literature on workflow modeling, several techniques are proposed to define and represent the structure of a business process (petri-nets, flow charts, etc.). In most cases, the method to be followed is imposed by the vendor of the workflow package [13]. Nevertheless, some general requirements can be put forward that are necessary to be able to model a process:

Requirement 1. Processes and activities need to be defined in a hierarchical manner. Process design typically requires a top down decomposition of high level processes into subprocesses down to atomic activities. It is the division of labor in the organization that determines the subdivision in sub-processes and activities. Activities and tasks might have a different meaning in different organizational theories. In the field of workflow modeling however, both terms are often used interchangeably and we will do the same in this paper.

Requirement 2. The modeling of dependencies between activities and between activities and agents is crucial. The main goal of a workflow system is in fact the automation or support of the co-ordination between activities and between activities and agents. Co-ordination can be defined as the management of dependencies [16]. In what way does one activity depend on the results of another activity? The modeling of dependencies constitutes the heart of workflow modeling. The existence of dependencies implies a certain order of execution. Some (sub)tasks cannot be performed before previous tasks have been completed; other tasks need to be executed in parallel, and so on.

Requirement 3. Agents are humans or computer applications that are assigned to roles. Also the interaction between activities and agents needs to be planned ahead. Agents can be human end-users or computer applications that perform activities. When human agents execute certain tasks, they might be assisted by computer applications to support them. Only when applications are directly coupled to the workflow system, they are considered as agents. When an application is invoked by the workflow system and when the application performs a certain activity without any intervention of the end-user, it is called an autonomous agent. An agent is called semi-autonomous when it is directly coupled to a workflow system, although an intervention of the end-user is still required. Agents are assigned to activities via the construction of roles. A role defines the responsibility for the performance of a (collection of) task(s)[14].

Requirement 4. The specification of the business process (or workflow) needs to be a persistent artefact able to control, supervise and record performed activities. The workflow process is specified as a model in a formal textual and/or visual language. This model specification is used whenever a new workflow instance needs to be created. Each time a workflow instance is created, the persistent workflow model is needed for controlling, supervising and recording the performed activities. Moreover, in order to monitor and improve performance, it is often also required to save the states of instantiated processes that have been enacted. Historical data regarding the actual course of processes can be useful and even necessary to improve the persistent process model.

The dependencies between activities and between activities and agents can be considered as the control logic of business processes. The functional part contains the necessary data and the applications that (partly) perform the activities (the non-human agents). The isolation of the control structure from the data and functional structure is a typical characteristic of workflow systems [25].

2.2 The Lack of a Business Process View in Object-Oriented Systems Development

Workflow systems and object-oriented technology have undoubtedly been some of the most important domains of interest of information technology over the past decade. Both domains however, have largely evolved independently, and not much research can be found in which workflow modeling principles and concepts have been applied to OO systems development. In object-oriented development, the primary emphasis is on the specification and development of the functional part of the information system, whereas the business process part is largely neglected or supposed to be given [e.g. 2, 3, 4, 6, 7, 11, 19, 23]. Although recently there is an increased support for the *software development* process and its workflows [9, 18], *business* process modeling is still treated in a fairly limited way.

In the first place, the top down decomposition of processes (requirement 1) is barely supported in object-oriented development. Functional decomposition, a vital concept from the structured programming world, is often considered as old-fashioned and ineffective by object-oriented developers [26]. One way to introduce some of the business process aspects in information systems analysis is the use of Use Cases [11, 2]. Use cases describe the functional requirements by identifying actors and scenarios of system usage by these actors. As such this technique is a valid candidate to model the interaction between a user and the system. The technique offers some possibilities for modularization by allowing use cases to "include" and "extend" other use cases, although this is not a functional decomposition as described in requirement 1 above. In the UML approach [11, 2], use cases are mainly a support for information system design: they are used for finding objects and determining the systems structure. More importantly, use cases are not intended to model the assignment of agents to activities and the co-ordination between activities (requirement 2) and can therefore not be considered as a workflow modeling technique. In addition the process logic is not designed to be implemented as an (persistent) application (requirement 4).

Because of their affinity with petri-nets, activity diagrams are much better suited for modeling activities and their dependencies. Although activity diagrams can be stored as persistent artefacts in a CASE-tool, they are not used to realize a workflow engine that controls, supervises and records the performed activities (requirement 4).

Finally, several other dynamic representations (like state transition diagrams and sequence diagrams) are created in the development phase. The process logic in this type of diagrams is however mainly relevant for the functional aspects of the application. In some cases, aspects of a business process can be found in this type of diagrams. However, such business process logic is not explicitly and separately implemented as described in requirement 4 above.

2.3 Advantages to Gain

A separation of concerns is a key element in keeping systems maintainable and adaptable. In current object-oriented system development practice, the organizational aspect of an information system is often not explicitly modeled. And when it is, it is not always taken as an important element in guiding design decisions. By integrating business modeling concepts into object-oriented modeling, the link between the services that an information system has to render and the organizational elements becomes more apparent. This can be an important help in designing more adaptable systems. In addition, when workflow elements are not modeled separately, they are often hidden in the procedural logic of class-methods. The explicit separation of workflow elements from process elements that are inherent to the domain or to the procedural logic of an implementation also allows for more adaptable systems. For example, sequence constraints on events that result from the business logic are part of the domain model (e.g. in a library, the return of a copy to the library must be preceded by a borrowing event). These type of sequence constraints are less likely to change over time than sequence constraints that are the result of workflow aspects

(e.g. if a member of the library does not show up after five reminders, set all the books (s)he borrowed to the state "lost").

2.4 Using Business Events as Bridging Concept

Events play a central role in the set of layers proposed in the next paragraphs. In most object-oriented approaches, events are subordinate to objects: they only serve as triggers for the execution of an object's method. In the approach proposed below, events are raised to the same level of importance as objects. Indeed, events are a fundamental part of the structure of experience [4]. Events are atomic units of action that represent things that happen in the world. Without events nothing would happen: they are the way information and objects come into existence (creating events), the way information and objects are modified (modifying events) and disappear from our universe of discourse (ending events). In the context of the business information system that gives the functional support for the workflow system, we make a difference between business events (also called real-world events) and information-system events such as keystrokes and mouse actions. The separation between these two types of events allows a more user-oriented and task-oriented view of information system design. Business events are those events that occur in the real world, even if there is no information system around. Information-system events are directly related to the presence of a computerized system. They are designed to allow the external user to register the occurrence of or invoke a real-world event. For example, the use of an ATM-machine to withdraw money from one's account will invoke the business event "*withdraw*" by means of several information-system events such as "*insert card", "enter PIN code", "enter amount",* and so on. Using events as a fundamental concept integrates well with the object-oriented approach as demonstrated in methods such as Syntropy [4], Catalysis [6], OO-SSADM [19] and MERODE [21, 23].

Business process modeling takes an action and process-oriented view on the domain. As a result, task and activities are easier to formulate in terms of business events than in terms of business objects (which are better for modeling structural aspects). From a business modeling perspective, only business events are of particular interest. Information system events such as keyboard actions and mouse clicks are modeled as elements in the information system, but are not relevant elements in a business process model.

As business events appear both in the functional part and in the business process part, they can serve as the bridging concept between workflow activities and information system design. The figures below represent a meta-model for the concepts used in the proposed system development approach. In a first step, business processes are modeled at a conceptual level by decomposing them down to the activity level and by indicating which business event each activity invokes. The meta-model for business process modeling only shows the BUSINESS PROCESS and WORKFLOW ACTIVITY classes. We assume however that the complete set of workflow concepts are modeled in an object-oriented way, such as for example in the $\text{TriGS}_{\text{flow}}$ model [14]. For the functional aspects, the considered domain is modeled at the conceptual level by iden-

tifying domain object classes and by indicating by which business event they are affected. As a result, business domain object interaction can be modeled by joint involvement in business event types, rather than by message passing. This makes the domain object interaction scheme more implementation independent and more adaptable to changed requirements. The effect of an event on a domain object class is recorded in a domain object class method. Fig. 1 shows a (simplified) meta-model relating the modeling concepts at this stage of the specification process. At the highest conceptual level, business events directly link business process modeling concepts to domain modeling concepts .

In a refining step the business processes and the business domain are analyzed in search for information system support. So, next to the description of the domain of interest in the domain model, we need a specification of the services (also called user functions) that the information system has to render to the prospective users. This part of the specification is closely related to the specification of the workflow model: it is the description of the functional support for the activities of the workflow model. The activities that have to be performed by agents can be further classified as manual, interactive or fully automated. Interactive and automated activities are realized by means of an information system service. In this refined model, the information system services interface the workflow system with the domain model by giving computerized support for the invocation of business events. Fig. 2. represents the meta model for this more detailed level of specifications.

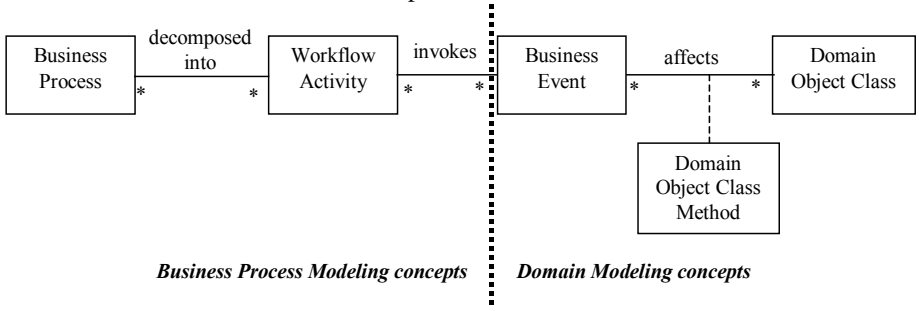

Fig. 1. Meta-model for conceptual modeling

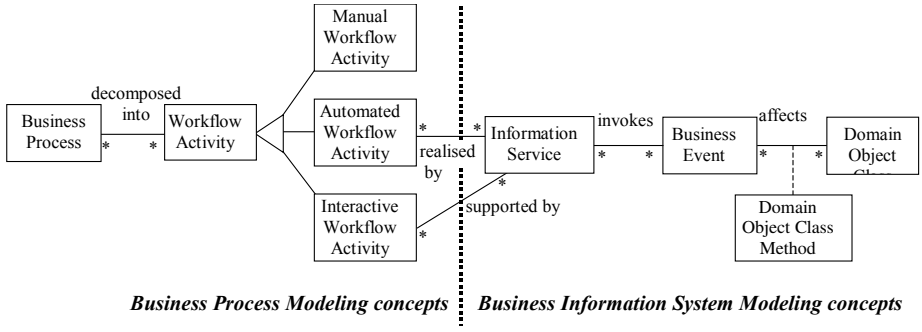

Fig. 2. Information systems modeling meta-model.

3. First Set of Basic Layers

The first set of layers are dictated by the separation of the workflow aspects from the functional support of tasks by information system services. Hence, at the highest level of abstraction, we have one layer for the workflow aspects and another layer for the information system aspects. A further refinement of the layers is obtained by applying the principle of model-driven development. Model-driven development is based on the idea that requirements should be captured in different models, according to their origin, as described in the work of Zachman [27, 24] and Maes[15]. Here we retain the separation of domain modeling from information system support modeling. Some specifications stem from fundamental business requirements (including business objects, business events as well as business constraints). This type of requirements is also valid if there is no information system. Other specifications are typically related to the presence of an information system. They describe the required functional support such as input facilities, generation of reports, EDI formatting, and so on. The specifications of the first type constitute a **business domain model** that contains the relevant domain knowledge for running a business. On top of this business domain model, a **information service model** is built as a set of input and output services, offering the desired information functionality to the users of the information system. Output services allow users to extract information from the business domain model, and present it in the right format on paper, on a workstation or in an electronic format. Input services allow users to register new or modified information that is relevant for the business.

The model-driven approach can also be applied to the workflow layer. The workflow domain model describes the essential concepts of business process modeling such as the concepts of agent, workflow activity, business process, task dependencies, worklists, and so on. It is by populating the workflow domain model with instances of agents, tasks, dependencies, business processes, etc. that the business processes of a particular organization are defined. At the same time the populated domain model is a persistent model of the organization's business processes (requirement 4). The workflow service layer describes the information system support offered by the workflow system such as facilities to view a worklist, to add a work-item to a work list, to pass on work items, to register the accomplishment of a task in a work list, to create new tasks, and so on, but also services that allow to control supervise and record performed activities (requirement 4). As a result, we obtain an architecture with 4 different layers (Fig. 3).

The dynamic aspects of each layer are triggered by four different kinds of events. Workflow-system events are information-system events related to the workflow system. They can for example be keyboard actions and mouse clicks to be captured by the workflow system and aiming at the invocation of a business-process event. Business-process events are real-world events that trigger the dynamic aspects of the workflow domain, such as the creation of a new task, assigning a person to a task, finishing a task and so on. They are related to the organizational aspects of the business. Information-system events trigger the dynamics of the information system. They will be used to invoke the execution of information system services that give

functional support for the tasks in the business process model. Finally, the business events are triggering the behavior of domain objects. They are the real-world events that constitute the dynamic part of a business.

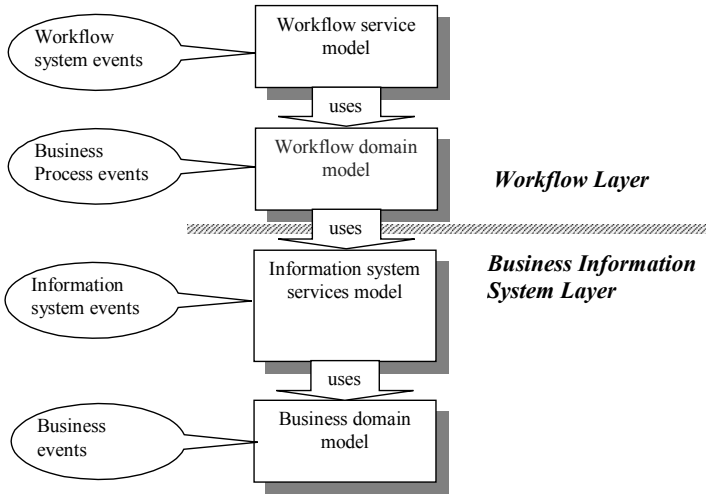

Fig. 3. Four basic layers.

The link between the workflow layer and the business information system layer is achieved by linking tasks in the workflow domain model with the supporting information system services. Execution of a task or work item means that either this service is automatically invoked by the workflow system (autonomous software agent) or that the user invokes the service him(her)self through the business information system user interface.

4. Refining the Layers

The four basic layers are each refined in two sub-layers. The business domain layer is subdivided in a business event layer and a business domain objects layer. As mentioned in section 2, it is assumed that business domain objects interact by being jointly involved in business event types rather than interacting through message passing. To realize this kind of interaction, it is assumed that events are broadcasted to objects. This means that when an event is invoked, each object that is involved in the event checks whether the constraints imposed by this object on events of that type are satisfied. If all the involved objects accept the event, all corresponding methods in the involved objects are executed simultaneously. This way of communication is similar to communication as defined in the process algebras CSP [10] and ACP [1] and has

been formalized in [5, 23]. Message passing is more similar to the CCS process algebra [17]. There exist various mechanisms for the implementation of such synchronous execution of methods. In the layered architecture proposed in this paper, we assume that there is an event handling mechanism that filters the incoming events by checking all the constraints this event must satisfy. If all constraints are satisfied, the event is broadcasted to the participating objects; if not it is rejected. In either case the invoking class is notified accordingly of the rejection, acceptation, and successful or unsuccessful execution of the event. For each type of business events, the event handling layer contains one class that is responsible for handling events of that type. This class will first check the validity of the event and, if appropriate, broadcast the event to all involved objects by means of the method 'broadcast'. This approach to domain object interaction enhances the adaptability of the domain model compared to a conventional approach were domain objects dispatch the event by sending messages to each other.

The information services layer can be further subdivided by separating user interface aspects from the transaction aspects. This separation is similar to the classical three tier architecture were control logic is also separated from user interface aspects. In this approach however, the control logic is partly in the transaction layer, partly in the business event layer, and partly in the methods of the business domain objects. User interface objects are responsible for all presentation aspects and for syntactical user input validation. Input transactions invoke one or more business events using parameter values received from the user interface objects. Output transactions query the set of domain objects to retrieve the requested information. The transaction layer can be used to group event invocations according to task requirements. Commit and roll-back features can also be implemented in the transaction layer.

To better illustrate the responsibilities of the different layers, we will exemplify objects in the four business information system layers by considering four examples of information system services for an order handling system. Let us assume that the domain model contains the four object types CUSTOMER, ORDER, ORDER LINE and PRODUCT, ORDER being existence dependent on CUSTOMER, and ORDER LINE being existence dependent on both ORDER and PRODUCT. The corresponding ER-schema is given in Fig. 4.

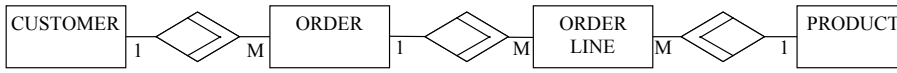

Fig. 4. ER-schema for the order handling system

Business event types are *create_customer, modify_customer, end_customer, create_order, modify_order, end_order, create_orderline, modify_orderline, end_orderline, cr_product, modify_product, end_product.* The object-event table (see Table 1) shows which object types are affected by which types of events and also indicates the type of involvement: C for creation, M for modification and E for terminating an object's life. For example, the *cr_orderline* creates a new occurrence of the class

ORDERLINE, modifies an occurrence of the class PRODUCT because it requires adjustment of the stock-level of the ordered product, modifies the state of the order to which it belongs and modifies the state of the customer of the order. Notice that Table 1 shows a maximal number of object-event involvements. If we do not want to record a state change in the customer object when an order line is added to one of his/her orders, it suffices to simply remove the corresponding object-event participation in the object-event table. Full details of how to construct such an object-event table and validate it against the data model and the behavioral model is beyond the scope of this paper but can be found in [21, 23].

Table 1. Object-event table for the order handling system

	CUSTOMER	ORDER	ORDERLINE	PRODUCT
create_customer	C			
modify_customer	M			
end_customer	E			
create_order	M	C		
modify_order	M	M		
end_order	M	E		
create_orderline	M	M	C	M
modify_orderline	M	M	M	M
end_orderline	M	M	E	M
create_product				C
modify_product				M
end_product				E

We consider four possible information system services: viewing a list of customers, creating a new customer, creating a new order with one or more order lines and deleting an order. For each of the services we identify relevant objects in each of the business information system layers and explain the interaction between these objects. Fig. 5 represents this graphically.

- *Viewing a list of existing customers*

  This will require an output transaction that queries the set of domain objects and presents the result of the query in a window. The execution of this transaction is invoked by means of user interface objects. Possibly, the user interface can allow to enter some search criteria the syntax of which is validated by the user interface objects. The result of the transaction is passed to user interface objects, responsible for presenting the resulting list of customers on screen.

- *Creating a new customer*

  This service requires an input transaction that will invoke the *create_customer* business event. The service is requested via the user interface layer, which is responsible for accepting user input of parameter values (e.g. customer name, ad-

dress, phone number) and for syntactical validation of these values (e.g. format of phone number, name must be alphabetical, …). The user interface objects pass the values to the transaction object Create New Customer in the transaction layer. The transaction object is responsible for creating an occurrence of the *create_customer* event type and invoking the execution of this event. In the event handling layer, the CREATE_CUSTOMER object is responsible for the further validation of the event against business rules (e.g. checking a uniqueness constraints on customer name and address) and for broadcasting the event to the involved objects in the business domain objects layer. In this case, the create_customer event will create a new occurrence of the CUSTOMER class.

- *Creating a new order with one or more order lines*

A possible implementation would allow users to enter the required data for an order together with a number of order lines on a single screen. These data are then passed to a multiple-event transaction object in the transaction layer. This example illustrates that a single transaction New Order can be further subdivided in sub-transactions such as Create Order and Add line to Order. The New Order transaction invokes the Create Order sub-transaction and one or more times the Add line to Order transaction. These sub-transactions will in their turn invoke a create_order event and the create_orderline events respectively. The integration of an additional service allowing to view a list of products that can be ordered (which would be a separate output transaction) must be done in the user interface layer.

- *Deleting an order chosen from a list of orders.*

This service requires the combination of an output transaction that generates a list of orders with an input transaction that invokes the *end_order* and *end_orderline* events. Commit and roll-back features can also be implemented in the transaction layer. In this service the transaction should, for example, only be committed if all order lines and the order were deleted successfully. If something went wrong during the transaction e.g. one of the *end_orderline* events was not invoked and broadcasted successfully, the roll-back feature allows to put all objects back to the state before the transaction was invoked.

The workflow domain layer and the workflow services layer are subdivided in exactly the same way. As a result, we obtain the four layers of Fig. 7 for the workflow layer.

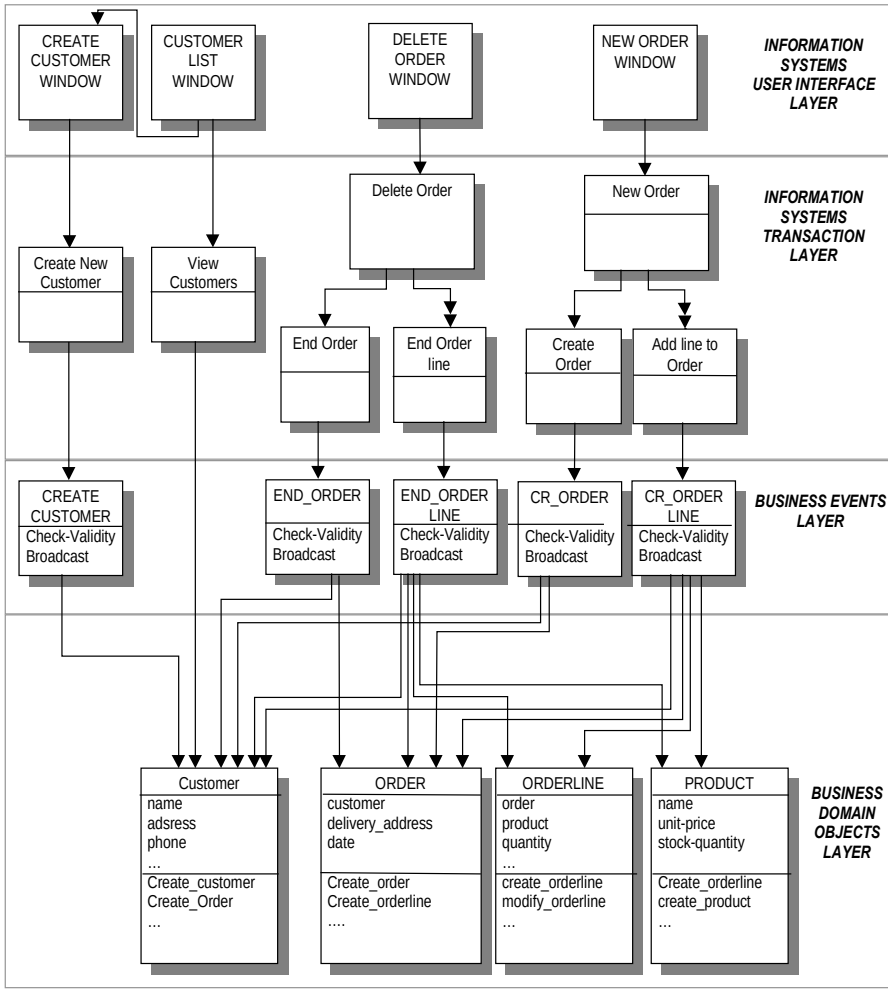

Fig. 5. Sublayers within the Business information system layer

5. Conclusion

This paper proposes a layered software specification architecture, guided by the principle of separation of concerns. The first set of layers was obtained by separating workflow aspects from functional support. By explicitly incorporating a workflow layer, we ensure that business process aspects are modeled separately. In the absence of such a layer, control aspects of business processes are often hidden in the objects that constitute the functional support for tasks, which is against the principle of separation of concerns.

These two layers were refined by separating domain modeling from information system modeling. Defining a domain model is an important requirements engineering step in the development of an information system: all business rules described in the

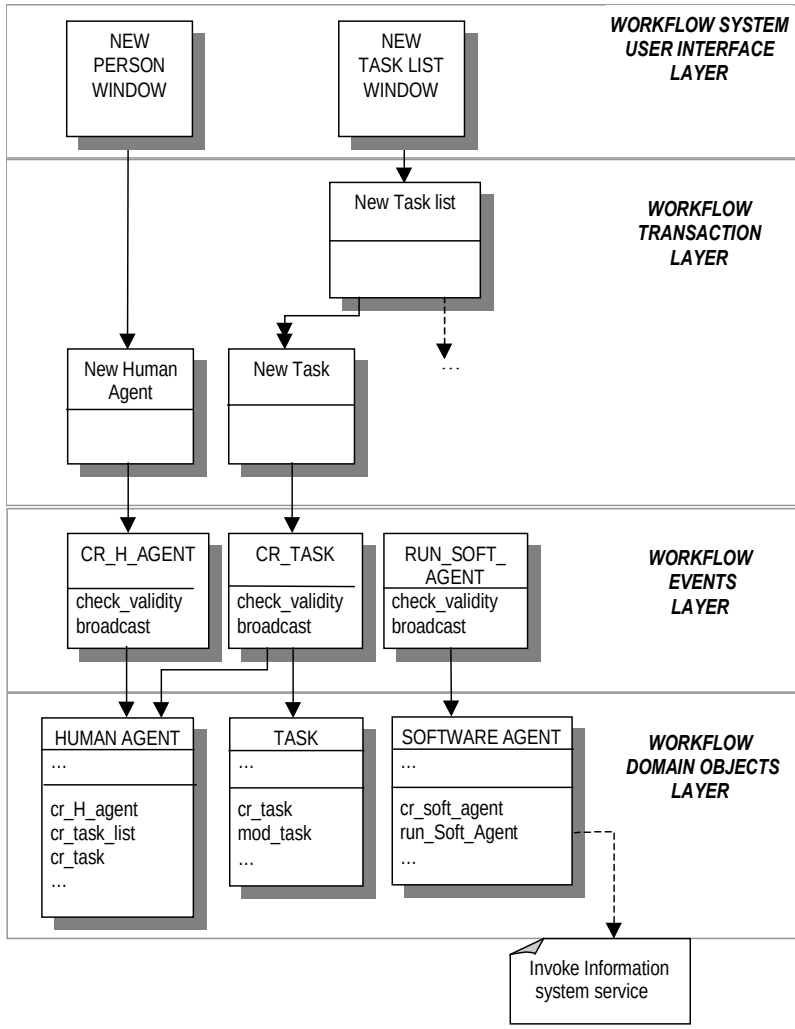

Fig. 6. Sublayers for the Workflow Layer

domain model must be supported by the information system. Methods such as JSD [12], OO-SSADM [19], Syntropy [4] and MERODE [23] even explicitly define domain modeling as a separate step in the development process. Interestingly, these method methods also recognize events as fundamental modeling concepts. The information system services are modeled as layer on top of the kernel layer constituted by the domain model. More importantly, information system services are independ-

ent units that can be plugged in and out the system without affecting the underlying domain layer. The services are glued together by means of the user interface layer. Again, this layer can be stripped off, without affecting the lower layers.

The layers for the business information system aspects are an integral part of the MERODE approach to software development which means that there is about 10 years of real-life experience with the business information systems layers. A survey amongst MERODE-users of this approach reveals that the separation of information system services aspects from domain modeling aspects indeed has a positive impact on modularity and hence on maintenance costs [22]. But according to the same survey, the separation of business knowledge from functional support has also other advantages: it results in a better understanding of the functioning of the business and it results in more transparent systems. The business information system layers can easily be compared to classical three tier architectures. Such architecture have for example, an application tier which contains the user interface aspects and part of the application logic, a domain tier and a persistent tier [7]. Jacobson [11] identifies three types of objects, presumably located in three corresponding tiers: entity objects (which constitute the domain model), control objects and user interface objects. The main difference with such three tier architecture is the identification of different types of control logic. In our approach the control logic is spread across different layers, according to the aspects it refers to. Business rules are stored as methods of domain classes or as general event constraints in the event handling layer. Application control aspects are located in the transaction and user interface layer. Control logic related to the organization of business process is stored in the definition of workflow domain objects, and finally, the workflow service layer captures control logic related to the use of a workflow system.

The model-driven approach which is one of the cornerstones of this paper is derived from the Zachman Information System Architecture [27, 24]. This architecture also contains a scope layer and a technology layer. The proposal of Maes [15] only retains the three lower layers, arguing that scope is a matter of ICT strategy development. As presented in this paper, the architecture does not yet include the third layer, that is to say, the technology aspects. In the Zachman and Sowa architecture [24] data, functionality, network, and other aspects are considered as orthogonal dimensions to the four basic dimensions (see Fig. 7). We expect that technology aspects have to be considered as an orthogonal dimension to the architecture of this paper. Indeed, different technology choices can be made for the realization of the different layers. For example, the business domain model and the workflow domain model can be realized with different database management systems and/or a different programming language. Also network aspects can be very different from one layer to another. One approach to deal with technology aspects is to combine code generation with the reuse of patterns and frameworks as proposed in [8].

	Data	Function	Network	People	Timing	Motivation
	what	*how*	*where*	*who*	*when*	*why*
SCOPE						
ENTERPRISE						
SYSTEM						
TECHNOLOGY						

Fig. 7. The extended Zachman framework for information systems architecture

References

1. Baeten, J.C.M., *Procesalgebra,* Kluwer programmatuurkunde, 1986
2. Booch, G., Rumbaugh, J., Jacobson, I., *The unified modeling language user guide*, Addison Wesley, 1999
3. Coleman, D. et al, *Object-oriented development: The FUSION method*, Prentice Hall, 1994
4. Cook, S., Daniels, J., *Designing object systems: object-oriented modeling with Syntropy,* Prentice Hall, 1994
5. Dedene G. Snoeck M. Formal deadlock elimination in an object oriented conceptual schema, *Data and Knowledge Engineering*, Vol. 15 (1995) 1-30.
6. D'Souza, D.F., Wills, A. C. Wills, Objects, Components and Frameworks with UML, The Catalysis Approach, Addison-Wesley, 1999, 785 pp..
7. Fowler, M., Analysis Patterns, Reusable Object Models, Addison Wesley Longman, 1997, 357 pp.
8. Goebl, W., Improving productivity in building Dat-Oriened Information Systems - Why Object Frameworks are not enough, Proc. of the 1998 Int'l Conf. On Object-Oriented Information Systems, Paris, 9-11 September, Springer, 1998.
9. Graham I., Henderson-Sellers B., Younessi H., The Open Process Specification (Open Series), Addison Wesley, 1997, 336 pp.
10. Hoare C. A. R., *Communicating Sequential Processes*, Prentice-Hall International, Series in Computer Science, 1985.
11. Jacobson, I., Christerson, M., Jonsson P. et al., *Object-Oriented Software Engineering, A use Case Driven Approach*, Addison Wesley, Rev. 4th pr., 1997.
12. Jackson, M.A., *System Development*, Prentice Hall, Englewood Cliffs, N.J., 1983.
13. Joosten, S., Werkstromen : een overzicht, in Informatie, jaargang 37, nr. 9, pp. 519-528.
14. Kappel, G., P. Lang, S. Rausch-Schott, & W. Retschitzegger, Workflow management based on objects, rules and roles, In Bulletin of the Technical Committee on Data Engineering, March 1995,18(1), pp. 11-18.
15. Maes, R., Dedene, G., *Reframing the Zachman Information System Architecture Framework*, Tinbergen Institute, discussion paper TI 96-32/2, 1996.
16. Malone, T. W., Crowston, K. The Interdisciplinary Study of Co-ordination, In ACM Computing Surveys, Vol. 26, No. 1, March 94, pp.87-119.
17. Milner, R., *A calculus of communicating systems,* Springer Berlin, Lecture Notes in Computer Science, 1980.
18. Rational Software Corporation, *The rational Unified Process*, http://www.rational.com/
19. Robinson, K., Berrisford, G., Object-oriented SSADM, Prentice Hall, 1994

20. Snoeck, M., Poels, G., Improving the Reuse Possibilities of the Behavioral Aspects of Object-Oriented Domain Models, In: Proc. 19th Int'l Conf. Conceptual Modeling (ER2000). Salt Lake City (2000)
21. Snoeck M., Dedene G. Existence Dependency: The key to semantic integrity between structural and behavioral aspects of object types, IEEE Transactions on Software Engineering , Vol. 24, No. 24, April 1998, pp.233-251
22. Snoeck M., Dedene G., Experiences with Object-Oriented Model-driven development, Proceedings of the STEP'97 conference, London, July 1997
23. Snoeck M., Dedene G., Verhelst M; Depuydt A.M., Object-oriented Enterprise Modeling with MERODE, Leuven University Press, 1999
24. Sowa J.F., Zachman J.A., Extending and formalizing the framework for information systems architecture, *IBM Systems Journal*, 31(3), 1992, 590-616.
25. Vaishnavi, V., Joosten, S. & B. Kuechhler, Representing Workflow Management systems with Smart Objects, 1997, 7 pp.
26. Wolber D., Reviving Functional Decomposition in Object-oriented Design, JOOP, October 1997, pp. 31-38
27. Zachman J.A., A framework for information systems architecture, *IBM Systems Journal*, 26(3), 1987, 276-292.