

Activity Model: A Declarative Approach for Capturing Communication Behaviour in Object-Oriented Databases

Ling LIU and Robert MEERSMAN

Infolab, Tilburg University, P.O.Box 90153, 5000 LE Tilburg, The Netherlands

Abstract

Communication behaviour represents the dynamic evolution and the cooperation of a group of objects in accomplishing a task. It is an important feature in object-oriented systems. This paper introduces an activity model for the declarative specification of such communication behaviour, including the temporal ordering of message exchanges within the object communications, and the behavioural relationships between activity executions. We develop two formal mechanisms: *activity specialization* and *activity aggregation* for abstract implementation of object communication in order to allow expressing complex behaviour in terms of simpler behaviour. Activities are seen as patterns of object communications, which explicitly capture the behavioural dependencies among cooperating objects, and proved to be an effective aid for implementing communication behaviour. We use first-order temporal logic for specification of communication constraints, and argue that the activity model as such provides an adequate computational framework for object communications, and strikes a better balance between the object communication paradigm and the object classification paradigm.

1 Introduction

The dynamic aspects of object-oriented systems have long been a central issue for modeling objects in the context of object-oriented programming (OOP) [cf.24], and more recently in the context of object-oriented databases(OODB) [cf.2,31]. The use of pre- and post-conditions for method specification, and the use of a trigger mechanism for the implementation of object interactions are commonly adopted for specifying behaviour.

In our view, the behaviour of objects can be classified into at least two categories. The first category is called *local behaviour* or intra-object behaviour. In performing local behaviour, only the components of objects and their attributive properties are affected, including those of subtypes. The second category may be called *behavioural composition* or inter-object behaviour, describing collections of independent objects cooperating to accomplish a task. It identifies a typical kind of communication behaviour, including the dynamic interactions between independent objects and the

communication constraints on those interactions. Recent literature [3,11,33] recognizes the importance of modeling communication behaviour in object-oriented systems.

Considering communication behaviour, several issues are of interest:

(1)Up to now, explicit specification and abstraction of communication behaviour are not well supported in most existing object-oriented models. Relationships are considered as attributive properties of objects and described via reference fields within the type definitions. It becomes then difficult to model the behaviour of relationships properly, in particular the dynamic constraints on the interactions between objects of independent types. This makes the communication behaviour difficult to understand, maintain, and debug. Any inconsistency between the static specification and dynamic specification of the problem domain, in general, will only be detected at the implementation stage. Furthermore, patterns of object communications within behavioural compositions are often repeated throughout a system with different participating objects. These patterns have limited reusability, because the semantics of grouping similar objects into classes and the semantics of capturing similar communication behaviours in terms of communication patterns are intertwined anyway. Thus, it becomes useful to provide some modeling construct for the declarative specification of patterns of object communications in object-oriented databases.

(2)Abstraction mechanisms, especially behavioural specialization and behavioural aggregation, are useful for defining new patterns of communication behaviour in terms of existing ones, and for supporting behaviour evolution of cooperating objects.

(3)In specifying object communication, it is necessary to define the temporal exchange ordering of the messages described within each type, the behavioural constraints and the temporal relationships between different communication behaviours, as well as the execution sequence of message exchanges among cooperating objects. Explicit specification of temporal knowledge of behaviour is actually indispensable for modeling object dynamics in non-procedural terms.

The objective of this paper is to introduce an activity model as a declarative specification formalism for modeling communication behaviour in object-oriented database design environment. The main contributions of our activity model

are the following. First, we generalize the concept of activities to describe communication behaviour between independent objects, and between different activities. The term "activity" is introduced as a semantically meaningful construct for explicit specification of communication behaviour, and for exploring the dynamic interactions between cooperating objects and the coordination of several activities in accomplishing a task. Second, we use the language of first-order temporal logic (FOTL) as the underlying formalism to capture the behavioural dependencies between cooperating objects. Thirdly, we formally develop two essential mechanisms — *activity specialization* and *activity aggregation* for abstract implementation of communication behaviours. The former allows us to model more specialized activities in terms of the existing activity specifications. The latter allows us to compose complex activities out of simpler activities. We shall show how activities enable us to create behavioural abstractions and to reuse the existing patterns of communications for defining new activities in order to avoid duplicate specifications, implementations and maintenance. Note that the notion of activity itself, of course, is not completely new [cf.7,12].

2 Background (Related Research)

Explicit specification and abstraction of communication behaviour seem not to have been addressed in a systematic and formal manner in the literature, although the research on dynamic data modeling, temporal databases, object migration, and the contract model, has made contributions to this subject in one way or the other. For instance, the transaction model [cf.25] developed by using the Active and Passive Component Modeling (ACM/PCM) methodology [cf.4] addresses the importance of modeling transactions prior to the completion of the static schema design. Process modeling, such as the event model [cf.9, 29] and the proposal in [22], suggests to structure process events hierarchically by communication links along which information is transmitted. Also, research on using adaptations of Petri nets [26] to model sequence of events, such as the scripting approach introduced by Borgida, Mylopoulos and Schmidt [6], uses extended Petri nets (scripts) to enforce dynamic integrity constraints and to define user interactions. The importance of specifying activity and behaviour in data modeling has also been emphasized in [13]. Recently Hall and Gupta [10] propose to model dynamic object migration from one subclass to another through transition modeling. Kappel and Schrefl [12] integrate the structure and behaviour representation by using object/behaviour diagrams, in order to model the life cycle of objects by stepwise refinement of their behaviour. There are also ongoing research in modeling the behavioural coordination of objects, such as the responsibility-driven

approach proposed by Wirfs-Brock and Wilkerson[33] and the contract model introduced by Helm, Holland, and Gangopadhyay in [11].

In our view, the constructs provided by the conventional transaction modeling formalism seems not quite suitable for high-level conceptualization of the communication behaviour, since in conventional transaction modeling, the behaviour of relationship objects can not be properly modeled, the knowledge of object evolution seems not fully exploited, and employed further for providing inference capabilities based on object history. To circumvent these problems, the temporal semantics of the universe of discourse should be explicitly addressed in the behaviour modeling, such as the sequence in which events or activities are happening, and the temporal ordering of message exchanges. The process model and the script model emphasize the state transition modeling, but provide no appropriate support for dynamic migration of entity objects from class to class, and no facilities for declarative specification of communication behaviours, such as the cooperation relationships among different processes, and the sequence of message exchanges between cooperating objects. The contract model and the responsibility-driven approach provide interesting ideas for modeling the mutual obligations of cooperating objects in accomplishing a task, but no formal semantics seem to be developed yet for abstract specification of communication behaviour.

3 The Activity Model

Given the universe of discourse (UoD), the activity model of the UoD consists of a nonempty set of activity patterns, describing the communication behaviours of cooperating objects in the UoD. The term "activity" will be introduced as a modeling construct for defining patterns of object communication. For brevity, in the current presentation we only focus on activity specification. A detailed formalism for object specification and classification may refer to [17,18].

3.1 First-order Temporal Logic

First-order temporal logic (FOTL) is a well-known extension of many-sorted first-order logic by adding modal operators for temporal inference. FOTL has been extensively used in the specification [cf.14], verification [cf.20], and synthesis [cf.21] of concurrent systems (e.g., concurrency and communicating processes). During the last decade, FOTL has also been introduced in the areas of information system design [cf.13,28], and specification and monitoring of database integrity constraints [cf. 15,16,27]. The logical system we use in this paper is linear and discrete with respect to the model of time [1].

The FOTL language consists of a countable set of FOTL terms and formulas; each is built up in the usual way by using predicates and function symbols, variables, standard quantifiers and connectives, plus the modal operators \Box ("always in the future"), \Diamond ("sometime in the future"), \bigcirc ("next"), *Until* ("until"), *Precedes* ("precedes"). If ϕ and ψ are formulas, then

- $\bigcirc\phi$ means " ϕ is true in the next state";
- $\Box\phi$ means " ϕ is always true from now on";
- $\Diamond\phi$ means " ϕ is true at some future time";
- ϕ *Until* ψ means " ϕ is true until ψ is true"; and
- ϕ *Precedes* ψ means " ϕ is true precedes ψ is true".

As an example, suppose we want to express the statement "if a property ψ holds for all individuals, then there must exist an individual in the next time point for which property ϕ holds" in FOTL. The result could be: $(\forall x)\Box\psi(x) \Rightarrow \bigcirc(\exists y)\phi(y)$. The formula $(\forall x)\Box\psi(x) \Rightarrow \bigcirc\Diamond\phi(x)$ is another example. It means that for any individual x , if $\psi(x)$ is true, then there should be a time point in the future such that $\phi(x)$ is true in the next state. For a detailed syntax of the FOTL see [19].

3.2 Activity Pattern

An activity pattern describes the communication protocol of a group of cooperating objects in accomplishing a task. We identify the following information necessary for specification of an activity pattern:

- The *type name* of each participating object in an activity (so called an *agent* of activity).
- The *set of messages* used by each agent when participating in an activity
- The *logic* of how objects play roles of being agents in an activity (so called *communication constraints* of the activity). For instance, which messages specified in the type/class schema are permissible for object participation in an activity, how the message exchanges are carried out, and what kind of mutual obligations the participating objects should follow.
- The *pre- and post-conditions* of an activity execution. For instance, what sorts of relationships between agents of an activity must hold, and how activities are invoked and what effect an activity might cause.
- The *composition* of a complex activity in terms of simpler activities and the behavioural characteristics of an activity, such as the relationships with other activities, and the execution sequence of its constituent activities.

In the sequel, we refer to the rules specifying relationships between the messages associated with each agent as *local constraints*. The rules on the message exchanges between two or more agents, and on the execution sequence of (constituent) activities, as well as on

the object-flow control between these activities are called *global constraints*. The preconditions of an activity ensure that each agent (participating object) has references (structural relationships) to the other agents and that the initial conditions are set up. The postconditions define what has to be satisfied after a successful activity execution. Failure of an activity execution will be reported by a system-supplied activity manager. The language for expressing constraints and preconditions is similar to FOTL. We describe each message in the form of message predicate *msg-name(sender, receiver, list-of-parameters)*. When a message is succeeded, the predicate is evaluated to be *true*. Otherwise it is *false*.

Definition 3.1 (activity pattern)

An activity pattern α is described by an eight-tuple $(N, \mathcal{AT}, \mathcal{PARA}, \mathcal{SUP}, \mathcal{AGG}, \mathcal{GIC}, \mathcal{PRE}, \mathcal{POST})$.

- N is the name of the activity pattern.
- \mathcal{AT} is a set of agents representing the objects participating in an activity of pattern α . Each agent is specified by a triple $\langle T, \mathcal{Msg}(\alpha, T), \mathcal{LIC}(\alpha, T) \rangle$ where T identifies the type name, $\mathcal{Msg}(\alpha, T)$ is the set of message predicates which are used for supporting the role of this type being an agent in the activity, and $\mathcal{LIC}(\alpha, T)$ is the set of rules (local constraints) on the exchange ordering of these messages.
- \mathcal{PARA} is a set of input and output parameters in an activity of pattern α , which may be component names or attributive property names of the participating agents.
- \mathcal{SUP} is a set of (super) activity names, in terms of which this activity is defined.
- \mathcal{AGG} contains a set of (constituent) activity names, of which this activity is composed.
- \mathcal{GIC} specifies the set of rules (global constraints) expressing the relationships between the activity and its agent objects, and between the constituent activities.
- \mathcal{PRE} is the set of preconditions and triggering conditions that initiate or hold at the start of the activity.
- \mathcal{POST} is the set of postconditions that hold after a successful completion of the activity execution. \square

Example: Consider as an application the communication between Airplane object and Control_tower object in accomplishing an activity TAKE_OFF. The activity pattern TAKE_OFF is specified in Fig3-1, which contains two agents of types Airplane and Control_tower respectively. The message predicates associated with each agent in the TAKE_OFF activity specification are well defined in the

```

activity TAKE_OFF
  agents  ct: Control_tower,
          pl: Airplane;
  in :    ctr-status(pl);
  out:    ctr-status(pl);
  Control_tower support
    {messages:
      request-take-off(self,ct,y), permission-take-off(ct,self,y),
      taken-off(self,ct,y);
      /* "self" here refers to an activity of pattern TAKE_OFF. /
    local constraints:
    LC11:  $(\forall y)\Box request-take-off(self,ct,y) \Rightarrow \bigcirc \Diamond permission-take-off(ct,self,y)$ ,
    LC12:  $(\forall y)permission-take-off(ct,self,y) \textit{Precedes} taken-off(self,ct,y)$ 
    };
  Airplane support
    {messages:
      com-take-off(self,pl),      change-ctr-status(self,pl,v),
      take-off-position(self,pl),  safe-check-ready(pl,self);
    local constraints:
    LC21:  $\Box com-take-off(self,pl) \Rightarrow \bigcirc \Diamond change-ctr-status(self,pl,v)$ ,
    LC22:  $safe-check-ready(pl,self) \textit{Precedes} take-off-position(self,pl)$ ,
    LC23:  $take-off-position(self,pl) \textit{Precedes} com-take-off(self,pl)$ 
    };
  global constraints:
  GC1:  $(\forall y)(y==pl \Rightarrow request-take-off(ct,self,y) \textit{Until} safe-check-ready(y,self))$ ,
  GC2:  $(\forall y)(y==pl \Rightarrow permission-take-off(ct,self,y) \textit{Precedes} take-off-position(self,y))$ ,
  GC3:  $\Box com-take-off(self,pl) \Rightarrow \bigcirc \Diamond taken-off(self,ct,pl)$ ,
  GC4:  $\Box taken-off(self,ct,pl) \Rightarrow \bigcirc \Diamond change-ctr-status(self,pl,"journey")$ ;
  precondition:
  PRE1:  $(\forall x)(\exists!y)Airplane(x)\wedge Control\_tower(y)\wedge has-take-off-ref(x,y)$ ,
  PRE2:  $(\exists x)(x==pl \wedge safe-check-ready(pl,self)\wedge ctr-status(x)="take-off")$ ;
  postcondition:
  POST1:  $(\forall x)(x==pl \wedge change-ctr-status(self,pl,"journey") \Rightarrow ctr-status(x)="journey")$ ;
end activity.

```

Fig3-1 An example of activity specification

type/class schema of this agent. Note that the set of messages defined in the type/class schema of Airplane objects includes not only the messages used for Airplane objects in participation of activity TAKE_OFF, but also the messages used for their participation in the other activities, such as JOURNEY_CONTROL, LANDING, EMERGENCY_LANDING, etc. Constraints associated with the TAKE_OFF activity concern the interactions between the activity itself and each of its agents, and the behavioural dependencies among these interactions.

Note that the local constraints associated with the agent *ct* (of type Control_tower) merely define the exchange ordering of all the messages, which support the Control_tower object being an agent in activity

TAKE_OFF. Similar to the local constraints associated with the agent *pl*. The exchange relationships between the messages of agent *ct* and the messages of agent *pl* are specified by the global constraints in activity TAKE_OFF. The precondition part identifies the legal situation for Control_tower objects and Airplane objects to participate in the activity. It states that at any time point, given an airplane object *x*, there must be one and only one control_tower object *y* (denoted as $\exists!y$) such that *x* has a (structural) reference to *y* and the function *ctr-status(x)* returns value "take-off". We provide a complete BNF syntax of the activity schema in [19]. \square

The introduction of activities as the coordinator for communication between objects has a number of

Let $CC(\alpha)$ be the set of the communication constraints of activity pattern α , and $CC_{\min}(\alpha)$ be the minimal set of the communication constraints of α .

$$CC(\text{TAKE_OFF}) = \{LC11, LC12, LC21, LC22, LC23, GC1, GC2, GC3, GC4\}.$$

$$CC_{\min}(\text{TAKE_OFF}) = \{LC11, LC23, GC1, GC2, GC3, GC4\}.$$

$$\begin{aligned} GC3 \wedge GC4 &\Rightarrow LC21, \\ GC2 \wedge LC23 \wedge GC3 &\Rightarrow LC12, \\ GC1 \wedge LC11 \wedge GC2 &\Rightarrow LC22. \end{aligned}$$

Fig3-2 Deleting the duplicated communication constraints LC12, LC21, LC22 from the activity specification in Fig3-1

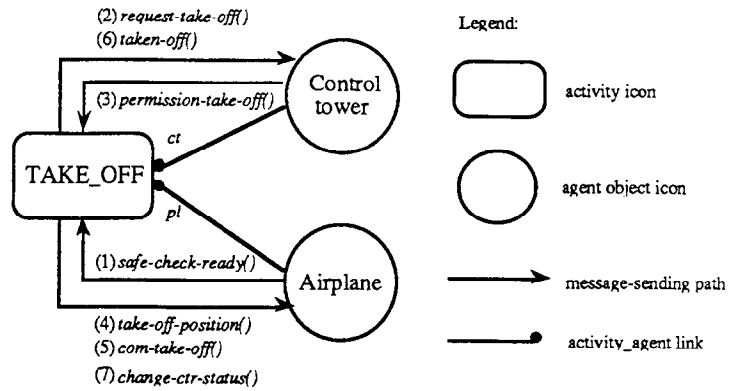


Fig3-3 The communication graph within TAKE_OFF activity.

advantages. It allows explicit specification of communication behaviour in one place, instead of having them hidden inside several type/class definitions or implicit in certain sequences of method calls. Also, it helps a designer to identify the standard communication protocols, independently from the implementation, by encouraging him to focus on the mutual obligations of the communication agents. As a result, reusability of the communication behaviour is increased.

3.3 Communication Path

In the activity model, we refer to the sequence of message exchanges implied in an activity pattern α as the *communication path* of α . Consider activity pattern TAKE_OFF in Fig3-1, the communication path implied in TAKE_OFF activity may easily be derived from the overall specification of the local and global constraints identified in Fig3-1. The deduction is based on the temporal inference rules [cf.1], and proceeds in two steps. Firstly, by eliminating duplicate constraint specification formulas, we get six FOTL formulas (i.e., LC11, LC23, GC1, GC2, GC3, GC4). They together form a minimal set of communication constraints that must hold as the invariants of activity TAKE_OFF. The conjunction of these six FOTL formulas is logically equivalent to the conjunction of all the (local and global) constraints specified in Fig3-1, because the others can easily be derived from these six formulas (Fig3-2). Secondly, based on this minimal set of constraint specifications, the message exchange sequence implied in the specification of activity TAKE_OFF can be generated simply by properly ordering all the messages involved in activity TAKE_OFF (Fig3-3).

Note that the concept of communication path (sequence of messages) is to some extent similar to (but not the same as) the concept of "trace" (sequence of events) introduced in Oblog [cf.29]. The requirements for safety (the conditions permitting event occurring), attribute valuation (the effect of events upon object attributes), and liveness

(the conditions for triggering events)[cf.30] in Oblog are also captured in the building blocks (constructs) of the activity model. For instance, we may specify safety requirement in the precondition, valuation in the postcondition, and liveness in the local and global constraints of activity specification. However, unlike the event model used in Oblog, our activity model allows to specify communication behaviour among multiple agents, including various communication constraints, at one place, rather than spreading them across over all the relevant object types. One obvious advantage is that we could model the mutual obligation and the interaction dependencies of the cooperating objects explicitly. Thus object communication behaviour can be fully captured and understood in a truly declarative way (i.e., at a higher level of conceptualization).

Definition 3.2 (communication graph)

Let $CG = \langle \mathcal{V}, \mathcal{E} \rangle$ be a graph with \mathcal{V} as the set of nodes and \mathcal{E} as the set of edges. CG is said to be a communication graph of an activity pattern α , if and only if \mathcal{V} consists of two kinds of nodes: communication modules and communication agents, and \mathcal{E} consists of two kinds of edges: the message-sending paths between activity of pattern α and its agents, and the activity_agent links indicating the agent objects of a given activity. \square

The communication graph of a TAKE_OFF activity is shown in Fig3-3.

Remarks: At this high level of specification, we are not concerned with the detailed methods of these messages. These details will be filled in during further design refinement or left until the implementation stage. Although activities are described separately from the type definitions of their agents, the activity implementation must finally be mapped to the methods defined in the type/class definitions of these agents. Besides, in an activity specification, for each agent object, its relationships with other cooperating agent objects are required to be specified in the type definition of this agent, in order to support its role of being

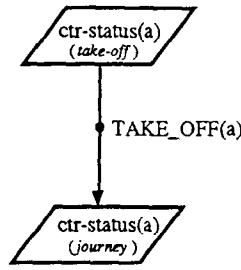


Fig3-4 An example of state transition within an activity

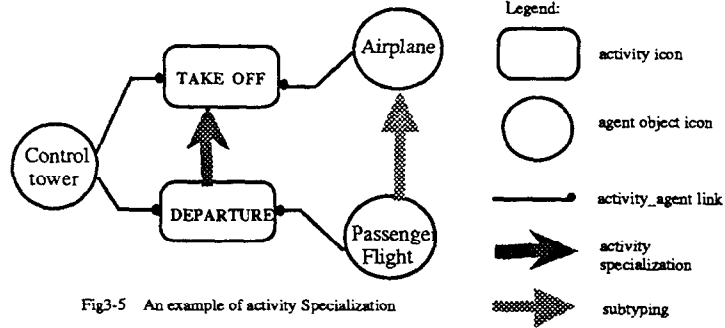


Fig3-5 An example of activity Specialization

agent in the activity. We also describe the state transition caused by an activity through the state transition graph associated with the activity pattern. This subject is somewhat beyond the focus of this paper. An example is shown in Fig3-4. It identifies that a success of any TAKE_OFF activity may cause a state transition of its agent Airplane object.

Given an activity pattern α , we refer to all individual activities of pattern α as *activity instances* of pattern α . For presentation convenience, in the sequel we may simply use "activity" to refer to activity pattern if no ambiguity results. An object may participate in several activities simultaneously. Also agents of an activity may be objects of the same type, of course.

3.4 Activity Specialization

Activity specialization is a behavioural abstraction mechanism, which allows us to define a more specialized pattern of communication behaviour in terms of existing patterns, for instance, by restricting behaviour to one or more specialized agent types instead, or through the addition of communication constraints. Behavioural specialization of relationships as such is quite common in object-oriented modeling practice.

Definition 3.3 (activity specialization)

Let

- $Agents(\alpha)$ be the set of agent type names in activity pattern α ,
- $Msg(T, \alpha)$ be the set of messages supporting the role of type T being an agent of α ,
- $GIC(\alpha)$ denote the conjunction of all the global constraints specified in activity pattern α ,
- $LIC(T, \alpha)$ denote the conjunction of all the local constraints associated with the agent of type T in activity pattern α ,
- $PRE(\alpha)$ be a set of pre-conditions that initiate an activity of pattern α , and
- $POST(\alpha)$ be a set of post-conditions that hold after a successful execution of activity of pattern α .

An activity pattern α is said to be a *behavioural specialization* of activity pattern β , if and only if (iff) the following conditions are verified.

- (i) $\forall S \in Agents(\beta), \exists T \in Agents(\alpha)$ such that $T=S \vee (\text{subtype_of}(T,S) \wedge (Msg(T,\alpha) \supseteq Msg(S,\beta) \wedge LIC(T,\alpha) \Rightarrow LIC(S,\beta)))$.
- (ii) $GIC(\alpha) \Rightarrow GIC(\beta)$.
- (iii) $\forall p \in PRE(\beta), \exists p' \in PRE(\alpha)$ such that $p=p'$, or p is overridden by p' .
- (iv) $\forall q \in POST(\beta), \exists q' \in POST(\alpha)$ such that $q=q'$, or q is overridden by q' .

We call α a *specialized activity pattern* and β a generic or *super-activity pattern*. \square

Remarks: The predicate "subtype_of(T,S)" holds iff type T is a subtype of type S . Condition (i) states that if activity pattern α is a *behavioural specialization* of activity pattern β , then for any type (say S) in $Agents(\beta)$, there is a corresponding type T in $Agents(\alpha)$ such that either T and S are same, or T is a subtype of S . In the later case, all messages, which are used for supporting type S being an agent in super-activity pattern β , are included in the message set for supporting type T being the corresponding agent in its specialized activity pattern α . Besides, the conjunction of all local constraints associated with agent type T in the specialized activity pattern α implies the conjunction of all local constraints associated with agent type S in the super-activity pattern β . Semantics of condition (ii), (iii), and (iv) are obvious.

By means of activity specialization, in defining activity α only the messages that are new and the constraints that need to be overridden are specified. All the other messages and constraints can be directly "inherited" from the super-activity β . We call this characteristic "*activity specialization inheritance*" in contrast with the concept of subclass inheritance in the object classification paradigm. The keyword "add support" is used in the activity model for specifying the new messages required in a specialized activity pattern.

```

activity - DEPARTURE
  agents   ct: Control_tower,
           pl: Passenger_plane;
  in :     ctr-status(pl);
  out:     ctr-status(pl);
  Behavioural specialization_of TAKE_OFF
    where Airplane = Passenger_plane;
  Control_tower add support {};
  Passenger_plane add support
  {messages:
    passengers-enter(self,pl);
    /* "self" in this example refers to an activity of pattern DEPARTURE. /
  local constraints:
    take-off-position(self,pl) Precedes passengers-enter(self,pl),
    passengers-enter(self,pl) Precedes com-take-off(self,pl)
  };
  global constraints:
  GC5: (∀y)(y==pl ⇒ passengers-enter(self,pl) until permission-take-off(ct,self,y));
  precondition:
  PRE1: (∀x)(∃!y)Passenger_plane(x)∧Control_tower(y)∧has-take-off-ref(x,y);
end activity.

```

Fig3-6 An example of activity specialization

Example: Consider activity DEPARTURE, which has Control_tower and Passenger_plane objects as agents. Passenger_plane is a subtype of Airplane. We may define the activity pattern DEPARTURE by behavioural specialization (refinement) of activity TAKE_OFF. Fig3-6 shows the definition of activity DEPARTURE.

Recall the activity specification of TAKE_OFF in Fig3-1, the activity DEPARTURE specified in Fig3-6 refines the activity TAKE_OFF in a number of ways. In activity DEPARTURE, the type of agent *pl* is replaced by Passenger_plane, and the additional message *passengers-enter()* is required. Besides, the method definition of *take-off-position()* overrides that in activity TAKE_OFF. Although a call to the message *take-off-position()* still leads to a call to *com-take-off()* as described in TAKE_OFF, the message *passengers-enter()*, which is specific to the agent of type Passenger_plane, will be called in between. Finally, more constraints may be involved in activity DEPARTURE than in TAKE_OFF. Note that the conjunction of all constraints required in activity DEPARTURE, however, implies the conjunction of all the constraints specified in activity TAKE_OFF. Furthermore, the precondition PRE1 specified in activity pattern TAKE_OFF is (partially) overridden in activity definition of DEPARTURE. But the agent of type Control_tower and the postconditions are not refined further in activity DEPARTURE. □

In contrast to the usual notions of specialization via subtyping or subclassification, the specialization of activities involves communication constraints on multiple agents, which may need to be refined.

Proposition 3.1

Let the predicate $act_specialization(\alpha, \beta)$ holds if and only if activity pattern α is a behavioural specialization of activity pattern β . Then

- (A1) $act_specialization(\alpha, \beta) \wedge act_specialization(\beta, \gamma) \Rightarrow act_specialization(\alpha, \gamma)$.
- (A2) $act_specialization(\alpha, \beta) \wedge act_specialization(\beta, \alpha) \Rightarrow \alpha = \beta$.

Proof

In terms of the activity specialization definition (recall Definition 3.3), and the transitive and antisymmetric characters of set inclusion and logical implication, as well as the partial ordering property of subtyping [18], we may immediately prove (A1) and (A2). □

3.5 Activity Aggregation

Intuitively, activities with the same or similar set of agents may "work together" in a more complex activity. The communication behaviour in such a complex activity with n agents may be described in terms of the communication behaviours related with some subsets of its agents and the relationships between these simpler communications. The task AIR_TRAFFIC_CONTROL, for instance, can be described as a complex activity, which consists of several constituent activities, such as TAKE_OFF, LANDING, JOURNEY_CONTROL, EMERGENCY_LANDING (see Fig3-7). Each of them has taken object of type Airplane and object of type Control_tower as their agents.

Constraints on the behavioural composition in the activity AIR_TRAFFIC_CONTROL include the constraints specified in each of its constituent activities, as well as the constraints on the relationships between these constituent activities (such as the sequence of message exchanges and the object-flow control between these activity executions). The sample specification of TAKE_OFF activity is given in Fig3-1. The formal specification of the others are omitted here due to the space limitation, and may refer to [19]. To avoid duplicate activity specification, we introduce *activity aggregation* mechanism to realize the aggregation abstraction of communication behaviour.

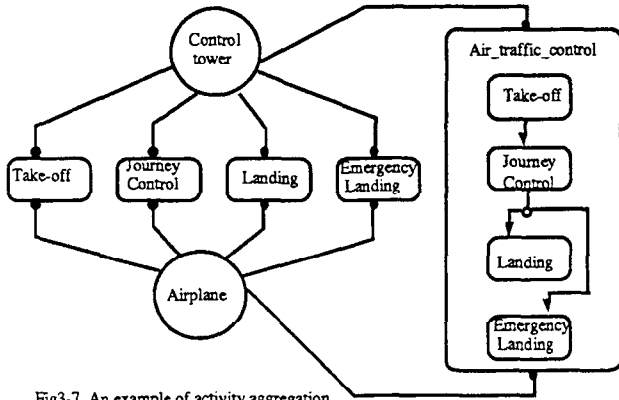


Fig3-7 An example of activity aggregation

Definition 3.4 (activity aggregation)

Let

- $Agents(\alpha)$ be the set of agents in activity pattern α ,
- $Msg(T, \alpha)$ be the set of messages supporting the role of type T being an agent of α ,
- $LIC(T, \alpha)$ denote the logical conjunction of all the local constraints associated with the agent of type T in activity pattern α ,
- $GIC(\alpha)$ denote the logical conjunction of all the global constraints in activity pattern α ,
- $PRE(\alpha)$ represent a set of pre-conditions that initiate an activity of pattern α , and
- $POST(\alpha)$ be a set of post-conditions that hold after a successful execution of activity of pattern α .

An activity pattern α is said to have a *behavioural aggregation* relationship with activity pattern β , if and only if (iff) the following conditions are verified.

- (i) $\forall S \in Agents(\beta), \exists T \in Agents(\alpha)$ such that
 $S = T \vee (\text{constituent_type_of}(S, T)$
 $\wedge (\forall i_t \in Msg(T, \alpha) \exists i_s \in Msg(S, \beta)$
 $(\square i_t \Rightarrow \square i_s) \wedge LIC(T, \alpha) \Rightarrow LIC(S, \beta)).$
- (ii) $GIC(\alpha) \Rightarrow GIC(\beta).$
- (iii) $PRE(\alpha) \supseteq PRE(\beta).$
- (iv) $POST(\alpha) \supseteq POST(\beta).$

We call α an *aggregate activity pattern* and β a *constituent activity pattern*. \square

Remarks:

In this definition, the predicate *constituent_type_of*(S, T) holds iff type T has an aggregation reference[18] to type S . Condition (i) states that if activity pattern α is a *behavioural aggregation* of activity pattern β , then for any type (say S) in $Agents(\beta)$, there is a corresponding type T in $Agents(\alpha)$ such that either T and S are same, or S is a constituent type of T . In the later case, for any message (say i_t) in the message set $Msg(T, \alpha)$, if the message i_t is sent out, then there should be a time point in the future such that, a message (say i_s) in the message set $Msg(S, \beta)$ will be invoked in the next state. Besides, the conjunction of all local constraints associated with agent type T in the aggregate activity pattern α implies the conjunction of all local constraints associated with agent type S in its constituent activity pattern β . Condition (ii) means that the conjunction of all global constraints in the aggregate activity pattern α implies the conjunction of all global constraints in its constituent activity pattern β . Condition (iii) actually imposes that the initiation (preconditions) of an aggregate activity includes (but not necessarily equal to) the union of the initiations of its constituent activities. Similarly, Condition (iv) requires that the effect (postconditions) of an aggregate activity includes the union of the effects of its constituent activities.

In terms of activity aggregation, in defining an aggregate activity α , only the messages and constraints that are new to the activity α , and that are not defined in the constituent activities of α need to be explicitly specified. The rest are simply "inherited" from the constituent activities of α . We call this characteristic "*activity aggregation inheritance*".

Note that a complex activity α may have some agents that are not participants in any of its constituent activities (say β_1, \dots, β_n). Also, for any agent of type T in a given constituent activity β_i ($i=1, \dots, n$), extra messages may be required for supporting its role of being also an agent in the aggregate activity α . We use the keyword "add" in the message specification of each agent, in order to identify such special agents as well as the new messages required by the aggregate activity α (see Fig3-8). Besides, the set of global constraints in activity α , i.e., $GIC(\alpha)$, may include both the global constraints in each constituent activity β_i ($i=1, \dots, n$) and the newly added global constraints in the activity pattern α .

Example: A sample specification of the complex activity AIR_TRAFFIC_CONTROL is given in Fig3-8. In specifying activity AIR_TRAFFIC_CONTROL, we only

```

activity AIR_TRAFFIC_CONTROL
agents  ct: Control_tower,
        pl: Passenger_plane;
Behavioural aggregation of
    TAKE_OFF, JOURNEY_CONTROL,
    LANDING, EMERGENCY_LANDING;
Control_tower add support
{messages
    start-take-off(self,ct),          start-journey(self,ct),
    start-landing(self,ct),          start-emergency-landing(self,ct),
    ... ; /* "self" in this example refers to an activity of pattern AIR_TRAFFIC_CONTROL. /
local constraints
    start-take-off(self,ct) Precedes start-journey(self,ct),
    start-journey(self,ct) Precedes start-emergency-landing(self,ct),
    start-journey(self,ct) Precedes start-landing(self,ct),
    ...};
Airplane add support {};
global constraints:
add
GC1:  $(\forall x) \square start\text{-take-off}(self,x) \Rightarrow \bigcirc \diamond (\exists a)(\exists y) TAKE\_OFF(a) \wedge request\text{-take-off}(a,x,y)$ ,
GC2:  $(\forall x)(\forall a) \square TAKE\_OFF(a) \wedge Agent\_of\_role(x,ct,a) \wedge taken\text{-off}(a,x) \Rightarrow \bigcirc \diamond start\text{-journey}(self,x)$ ,
GC3:  $(\forall x)(\forall a) \square JOURNEY\_CONTROL(a) \wedge Agent\_of\_role(x,ct,a) \wedge request\text{-emergency-landing}(a,x) \Rightarrow \bigcirc \diamond start\text{-emergency-landing}(self,x)$ ,
GC4:  $(\forall x)(\forall a) \square JOURNEY\_CONTROL(a) \wedge Agent\_of\_role(x,ct,a) \wedge finish\text{-journey}(a,x) \Rightarrow \bigcirc \diamond start\text{-landing}(self,x)$ ,
GC5:  $(\forall x) \square Agent\_of\_role(x,ct,self) \wedge start\text{-emergency-landing}(self,x) \Rightarrow \bigcirc \diamond (\exists a) EMERGENCY\_LANDING(a) \wedge Agent\_of\_role(x,ct,a) \wedge permit\text{-emergency-landing}(x,a)$ ,
GC6:  $(\forall x) \square ((\exists a) JOURNEY\_CONTROL(a) \wedge Agent\_of\_role(x,pl,a) \wedge out\text{-of-fuel}(x,a) \Rightarrow \bigcirc \diamond (\exists b)(\exists y) EMERGENCY\_LANDING(b) \wedge Agent\_of\_role(y,ct,b) \wedge permit\text{-emergency-landing}(y,b,x)$ ,
GC7:  $(\forall x) \square ((\exists a) TAKE\_OFF(a) \wedge Agent\_of\_role(x,pl,a) \Rightarrow \bigcirc \diamond (\exists b)((LANDING(b) \vee EMERGENCY\_LANDING(b)) \wedge Agent\_of\_role(x,pl,b))$ ,
GC8:  $(\forall x) \square ((\exists a) LANDING(a) \wedge Agent\_of\_role(x,pl,a) \Rightarrow \bigcirc \diamond (\neg \exists b)(EMERGENCY\_LANDING(b) \wedge Agent\_of\_role(x,pl,b))$ ,
.....
end activity.

```

Fig3-8 An example of activity aggregation

need to define the messages and the constraints that are not described in any of its constituent activities. These additional constraints will identify the object-flow control in a sequence of activity executions and the temporal ordering of message exchanges among the involved activities. Consider the global constraints in Fig3-8, GC1 and GC2 specify the conditions which must hold between the messages in activity AIR_TRAFFIC_CONTROL and the messages in activity TAKE_OFF. GC3 and GC4 identify the conditions which must hold between the messages in JOURNEY_CONTROL activity and the messages in activity AIR_TRAFFIC_CONTROL. GC5 states the

conditions which must hold between messages in EMERGENCY_LANDING activity and messages in AIR_TRAFFIC_CONTROL activity. GC6 states some dynamic relationship between activities JOURNEY_CONTROL and EMERGENCY_LANDING. GC7 and GC8 specify the execution sequence which must apply to activities TAKE_OFF, EMERGENCY_LANDING or LANDING for the same flow of participating objects. It means that once an airplane object participates in activity TAKE_OFF, then at some time in the future, it must also participate in either activity LANDING or activity EMERGENCY_LANDING, but the LANDING activity and

```

activity Passenger_plane_Traffic_Control
  agents   ct: Control_tower,
            pl: Passenger_plane;
  Behavioural specialization_of AIR_TRAFFIC_CONTROL
    where   Airplane = Passenger_plane,
            TAKE_OFF = DEPARTURE,
            LANDING = ARRIVAL,
            ...
            add constituent activity MIDWAY_STOP;
  .....
end activity.

```

Fig3-9 An example of combining activity specialization with activity aggregation

the EMERGENCY_LANDING activity can never happen at the same time to the same object. \square

Proposition 3.2

Let the predicate $\text{act_aggregation}(\alpha, \beta)$ hold if and only if activity pattern α has an activity aggregation relationship with activity pattern β . Then

(A3) $\text{act_aggregation}(\alpha, \beta) \wedge \text{act_aggregation}(\beta, \gamma)$
 $\Rightarrow \text{act_aggregation}(\alpha, \gamma)$.

(A4) $\text{act_aggregation}(\alpha, \beta) \wedge \text{act_aggregation}(\beta, \alpha)$
 $\Rightarrow \alpha = \beta$.

Proof

The proof follows immediately from Definition 3.4, the laws of set inclusion and logical implication, and the acyclicity [18] of the aggregation type hierarchy. \square

So far we have defined two abstraction mechanisms for activity composition. They are activity specialization and activity aggregation. In fact, these two mechanisms may be combined within the activity model. The complete specification syntax of activity aggregation and activity specialization can be found in [19].

Example: Fig3-9 provides an illustration. It defines Passenger_plane_Traffic_Control activity, which is a behavioural specialization of the aggregate activity AIR_TRAFFIC_CONTROL.

Passenger_plane_Traffic_Control activity (PTC for short) refines AIR_TRAFFIC_CONTROL activity (ATC for short) in a number of ways. First, the type of the agent Passenger_plane in PTC is a subtype of the corresponding agent type Airplane in ATC. Second, the constituent activities DEPARTURE and ARRIVAL in PTC are specialized activity patterns of TAKE_OFF and LANDING in ATC respectively. Thirdly, activity PTC has more constituent activity patterns than ATC, e.g., MIDWAY_STOP which is not included in activity pattern ATC. To specify the newly added constituent activity patterns such as MIDWAY_STOP, we introduce the


keyword "add constituent activity" in the definition of activity specialization (see Fig3-9). \square

We argue that by combining activity aggregation with activity specialization, a powerful mechanism for abstract implementation of communication behaviour in object-oriented databases results. As a consequence, the specification and implementation code of communication behaviour can be drastically simplified. Thus database programs, especially those for implementing object communication behaviour, may substantially be better understood and reused.


4 Activities v.s. Entity Objects

Activities and entity objects have a number of characteristics in common. For example, in order to distinctly identify a particular activity, we need to assign each activity with a unique activity identifier (we may use, e.g., the pattern name of an activity and the object identities of its agents as the signature of this particular activity). Also like entity objects, activities may have relationships with each other, and be composed of other activities. Finally, the behavioural relationships between activities are also realized through message exchanges between activities and their participating objects.

However, activities differ conceptually from entity objects in a number of ways. Activities are dynamic "objects" which come into existence and disappear according to a set of transition rules, whereas entities, relatively speaking, are static object types, of which instances are created and deleted according to application specifications. Activities are introduced to explicitly specify the dynamic structure of objects and the behaviour of relationships between independent objects, and to model complex communication behaviour in terms of simpler behaviours. Whereas in the class/type definition, only the static structure and the behavioural interaction between objects of the same type or between objects of dependent types are explicitly identified. The behavioural interactions between

 **Sequentialization** (symbol: \vdash)

A sequence of activities means that these activities must execute in the given order. Suppose activities a, b, c are to be executed one after the other in the order a before b and b before c . This can be expressed by " $a \vdash b \vdash c$ ".

 **Selection** (symbol: $[, \dots]$)

A selection from a set of activities permits only one to activate. We may use " $[a, b, c]$ " to denote that the execution of activities a, b , and c are going to be selectively synchronized.

 **Repetition** (symbol: $*$ or $+$)

It allows specification of synchronization to be repeated. a^* means that activity a can be repeated zero or more times, and a^+ means that activity a can be repeated one or more times.

 **Concurrency** (symbol: \parallel)

It allows the specification of concurrent synchronization. For instance, " $(a \parallel b) \vdash c$ " denotes that activities a and b can be executed concurrently before activity c is invoked.

Fig5-3 Four fundamental synchronization schemes

independent objects are either hidden inside several type/class definitions or implicit in certain sequences of method calls. Besides, constraints associated to a particular activity make it possible to describe the dynamic interaction and the sequence of message exchanges between objects of independent types and between activities. While constraints associated with an object type can at most specify the static or simpler relationships with the objects of other types, and the temporal order of the messages associated with the objects of the same type. Finally, an activity can access the participating objects only through messages (thus properties) which are used by these objects for supporting their roles of being agents in the activity. To manipulate other properties of the objects, some other activities and methods must be invoked for sending appropriate messages to them. A complex object may participate in multiple activities; this may result in a large number of methods in its public (type) interface. The activity specification actually *factors* this large interface into meaningfully related subsets. This is also a principal reason for stating that the construct "activity" is a valid abstraction mechanism for modeling object communication dynamics.

5 Activity Model v.s. Event Model

In the last decade many researchers have studied event modeling and process modeling for specification of object dynamics. In this section we briefly discuss the similarities and the subtle differences between the activity model and the event model.

The similarities of the activity model and the event model are the following.

- (i) They both use the pre- and post-condition to specify the triggering and initiation of object behaviours;
- (ii) They both differ from the process model in the way that the process model has more interest on the operational aspect of each action and the sequence of actions but much less attention on how a process is

initiated and triggered.

- (iii) Like processes, both of them supports the fundamental synchronization schemes (see Fig5-3).

As the messages used in an activity may be synchronized sequentially or concurrently, these synchronization schemes can also be used to express the synchronization structure of message exchanges.

Example:

- (1) Given two agents ct, pl , the synchronization structure of activity executions implied in Fig3-8 can be described by

TAKE_OFF \vdash JOURNEY
 \vdash [LANDING, EMERGENCY_LANDING].

- (2) Suppose we add messages $change-speed(self,pl)$, $change-direction(self,pl)$, and $change-altitude(self,pl)$ into the message set $Msg(TAKE_OFF, Airplane)$ in Fig3-1. They will be triggered after a TAKE_OFF activity have received message $com-take-off(self,y)$ from Airplane object and sent out message $taken-off(self,ct,pl)$ to Control-tower object. The synchronization structure of message exchanges implied in TAKE_OFF activity is described as follows:

$safe-check-ready(y,self) \vdash request-take-off(ct,self)^+$
 $\vdash permission-take-off(ct,self,y)$
 $\vdash take-off-position(self,y) \vdash com-take-off(self,y)$
 $\vdash (taken-off(self,ct,pl) \parallel change-direction(self,pl)^*$
 $\parallel change-speed(self,pl)^* \parallel change-altitude(self,pl)^*)$
 $\vdash change-ctr-status(self,pl, "journey").$

However, the activity model conceptually differs from the event model. The activity model focuses more on the object communications and the relationships between sequence of communications, but emphasizes less on the result and outcome of a process and on what actions are triggered to react messages. Whereas the event model

concerns more with the outcome of a process and the actions undertaken within the process, but emphasizes less on how objects communicate with each other during a process and the relationships between series of actions, hereby usually lacking the "time dimension" inside an event. As a result, the mutual obligation and interaction dependencies implied in the communication and coordination of multiple agents are missing too.

6 Conclusion

We have presented an activity model for the abstraction, aggregation, and specialization of communication behaviours. Unlike most of the conventional object-oriented models, it advocates the explicit specification of communication behaviours in one place, instead of having them hidden inside several type/class definitions or implicit in certain sequences of method calls. We argue that this approach may result in substantial benefits, including declarative specification of communication behaviour at a high level of conceptualization, enabling of temporal inference, more effective integration of high level dynamic modeling with structural modeling, and a conceptual level support for lower level database transaction design, and consequently a higher degree of reusability and extensibility of the object-oriented design.

Through the support of abstraction mechanisms for classification and composition of activities, the existing activity patterns can be combined to define complex activity patterns, and the generic activity patterns can be reused by different object types through activity specialization. Activity patterns can also be reused together with their agent types. Besides, using the activity model for specification of communication behaviour, allows to reduce the number of subtypes or subclasses, and to minimize the need for adding subtypes/subclasses and for object migration from subclass to subclass.

Our main contribution in this paper is the formal introduction of behavioural abstraction mechanisms: activity aggregation and activity specialization for modeling object communication behaviour. Thus in defining aggregate or specialized activity, only new behavioural characteristics need to be specified, the others can all be "reused" or "inherited" from the existing activity specifications. It is also possible to override some behaviour characteristics such as replacing agent types by their subtypes, enforcing stronger constraints on initiating an activity, the ordering of message exchange, and the execution sequence of activities. The concept of activity aggregation and in particular its combination with activity specialization, to our knowledge, have so far not been reported in the literature.

Another contribution of the framework presented herein is an attempt to integrate the database object

classification structure with the communication structure. Due to the space limitation, the demonstration of how such an integration would enhance the reusability of database programs, and how it contribute to an object-oriented design environment is omitted in the current presentation, and may refer to [19]. Clearly, classes enable component-level and framework-level reuse, and the activity model enables communication-based reuse. We quite agree with the statement made by Wegner in [32] that "*providing a computational framework for classification is comparable in its practical importance and research interest to providing a computational framework for communication*".

The work on the present activity model in the areas of formal theory, design methodology and tool support continues. Studies on the formal theory includes the theoretical foundation of the activity model, the consistency checking of the activity schema with the type/class definitions, and the expressive power of the activity schema as a knowledge representation language in the domain of modeling object dynamics. We are also interested in developing a transformation engine that maps the declarative activity specification into the procedural specification of database transactions or stored procedures. The issues of monitoring activity constraints or invariants and the temporal pattern of communication processes, such as homogeneous activity, majoritative activity, occasional activity, and holistic activity, merit attention.

Acknowledgement: We thank Alex C. Uuldriks for the discussion and the effort towards implementation.

References

- [1] Abadi M. and Manna Z., Nonclausal deduction in first-order temporal logic. In: Journal of the ACM Vol.37, No.2 (April, 1990) pp279-317.
- [2] Abiteboul S., Kanellakis P.C., and Waller E., Method schemas. In: PODS (1990) pp16-27.
- [3] Booch Grady, Object-oriented design. Benjamin/Cummings Publishing Co., (1991).
- [4] Brodie M., Mylopoulos J., and Schmidt J.W. (ed.) On Conceptual Modeling, Perspectives from Artificial Intelligence, Databases, and Programming Languages. (Springer Verlag, 1984)
- [5] Brodie M. and Ridjanovic D. On design and specification of database transactions. In: [4] pp277-306.
- [6] Borgida A., and Mylopoulos J., and Wong H.K.T., Generalization/specialization as a basis for software specification. In: [4] pp87-114.
- [7] Chung L., et.al, From information system requirements to designs: a mapping framework. In: Info. Systems Vol.16, No.4 (1991) pp429-461.
- [8] Greenspan S., Borgida A., and Mylopoulos J., A requirements modeling language and its logic. In: On

- Knowledge Base Management Systems (eds. by M. Brodie and J. Mylopoulos) (1985) pp471-502.
- [9] King R. and Mcleod D., A unified model and methodology for conceptual database design. In: [4] pp313-327.
- [10] Hall G. and Gupta R., Modeling Transition. In: IEEE ICDE'91 (Kobe, 1991) pp540-549.
- [11] Helm R., Holland I.M. and Gangopadhyay D., Contracts: specifying behavioural compositions in object-oriented systems. In: OOPSLA (1990) pp169-180
- [12] Kappel G. and Schrefl M., Object/behaviour diagrams. In: Proc. IEEE ICDE'91 (1991) pp530-539.
- [13] Kung C.H. and Solvberg A., Activity and behaviour modeling. In: Info. Sys. Design methodologies: Improving the Practice. Elsevier, Amsterdam (1986)
- [14] Lamport L., Specifying concurrent program modules. In: ACM Trans. Prog. Lang. Syst. (1983) pp190-222.
- [15] Lipeck U.W., Stepwise specification of dynamic database behaviour. In: VLDB (1986) pp387-397.
- [16] Lipeck U.W. and Saake G., Monitoring dynamic integrity constraints based on temporal logic. In: Info. Systems. Vol.12, No.3 (1987) pp255-269.
- [17] Liu L. and Dijkstra J., Object-oriented Specification Model and Its Tool Support. In: Proc. Int. Conf. on Management of Data (Bangalore, 1992).
- [18] Liu L., Exploring semantics in aggregation hierarchies for object-oriented databases. In: Proc. IEEE ICDE (Phoenix, Arizona 1992) pp116-125
- [19] Liu L. and Meersman R. Activity Model: a declarative approach for capturing communication behaviour in object-oriented databases. KUB/Infolab Research Report, (Nov. 1991) pp1-65
- [20] Manna Z. and Pnueli A., Verification of concurrent programs: the temporal framework. In: Correctness Problem in Computer Science (ed. R.S.Boyer, J.S.Moore) (Academic Press, 1981) 215-273
- [21] Manna Z. and Wolper P., Synthesis of communicating processes from temporal logic specifications. In: ACM Trans. Prog. Lang. Syst. Vol.6, No.1 (1984) pp68-93.
- [22] Markowitz V.M., Representing processes in the extended entity-relationship model. In: Proc. IEEE ICDE (1990) pp103-110
- [23] Meersman R., Sernadas C., Fiadeiro J., Sernadas A., Proof-theoretic conceptual modeling: the NIAM case study. In: Proc. of working conf. on Fundamental Concepts for Information Systems. E.F.Falkenberg (ed.) (North Holland, 1989) pp1-30
- [24] Meyer B. Object-oriented Software Construction. PrenticeHall International, 1988.
- [25] Ngu A.H.H., Transaction modeling. In: Proc. IEEE ICDE (1989) pp234-241
- [26] Peterson J.L., Petri net theory and the modeling of systems. Prentice-Hall, Inc., 1981.
- [27] Saake G., Descriptive specification of database object behaviour. In: DKE 6 (1991) pp47-73.
- [28] Sernadas A, Temporal aspects of logical procedure definition. In: Info. Syst., Vol.5 (1980) pp167-187.
- [29] Sernadas A, What is an object after all?. In: Object-oriented Databases: Analysis, Design & Construction (DS-4), Meersman R., Kent W., and Khosla S. (ed.) (North-Holland, 1991) pp39-70.
- [30] Fiadeiro J., Sernadas C., Maibaum T., and Saake G., Proof-theoretic semantics of object-oriented specification constructs. In: Object-oriented Databases: Analysis, Design & Construction (DS-4), Meersman R., Kent W., and Khosla S. (ed.) (North-Holland, 1991) pp243-284.
- [31] Tjoa A.M. and Wagner R.R. Comparison of dynamic aspects in semantic data modeling. In: Proc. Int. Conf. Applied informatics (Acta Press, Zurich 1985).
- [32] Wegner P., Classification in object-oriented systems. In: ACM SIGPLAN (1986) pp173-182
- [33] Wirfs-Brock R. and Wilkerson B., Object-oriented design: a responsibility-driven approach. In: ACM OOPSLA (NewOrleans, LA, 1989) pp71-76.