Six Spaces for Global Information Systems Design

FALQUET, Gilles, et al.

Abstract

We introduce a model for information systems specification which covers the spaces of data, semantic contexts, events, objects life-cycle, integrity constraints, and operating periods. For each one of these space we define a structure and its interpretation that allow to specify this particular aspect of the system. In this approach data and event structures are independent. It is the object life-cycle structure that binds processes to objects on which they operate. Semantic contexts make process and integrity constraints independent from the details of the data structure. The definition of operating period determine which integrity constraints are active and which processes are allowed to execute at a given time. When put together, these specifications form a global information system specification that can be checked for consistency and completeness and that can be used to predict global system’s behavior.

Reference


Available at: http://archive-ouverte.unige.ch/unige:46593

Disclaimer: layout of this document may differ from the published version.
Six Spaces for Global Information Systems Design

Th. Estier, G. Falquet, J. Guyot, M. Léonard

Centre universitaire d’informatique, Université de Genève, Rue du Lac 12, CH-1207 Genève, Switzerland

Abstract

We introduce a model for information systems specification which covers the spaces of data, semantic contexts, events, objects life-cycle, integrity constraints, and operating periods. For each one of these space we define a structure and its interpretation that allow to specify this particular aspect of the system. In this approach data and event structures are independent. It is the object life-cycle structure that binds processes to objects on which they operate. Semantic contexts make process and integrity constraints independent from the details of the data structure. The definition of operating period determine which integrity constraints are active and which processes are allowed to execute at a given time. When put together, these specifications form a global information system specification that can be checked for consistency and completeness and that can be used to predict global system’s behavior.

1 Introduction

Our aim is the definition of a conceptual model which integrates aspects of application fields which have been left out at the conceptual level. This should improve quality of the system specification, especially in its ability to predict and validate the behavior of the system.

In this paper we describe a global conceptual model of information systems. Although the notion of global behavior has been extensively studied in the field of real-time systems, it has attracted less attention in the information systems area and only a few models, such as IDA [18] and Rubis [3], consider this point. Our model is the result of research into specific domains like process modeling, data modeling, query semantics, integrity rules validation, and into design process.

In the first part of this paper we introduce six design spaces of an application field. In the second part, we describe our conceptual model which exactly covers these six spaces. For each space, we give structural aspects of the model, related theoretical issues, and an interpretation with its constraints. In the next part we study interactions between these spaces.
In the fourth part we examine how existing models include some of the described spaces in their approaches and show the benefits of the logical and behavioral independence introduced by our model.

Finally we describe how this model solves problems appearing during the design or the operation of the information system. We show how this model can be (and has been) implemented on top of existing systems.

2 Six spaces of an application field

The proposed model covers six different spaces of an application field.

1. Data

This space is the set of objects that constitute the current state of the system. This space is structured in values, domains, types, classes, and a set of semantic links between classes. The interpretation of a data-structure is built over values compatible with the structure constraints. For example, in the relational model, the interpretation of a relation schema will be a relation (a set of tuples) whose elements belong to the attributes domains.

2. Events

Events are significant phenomenons of the application field, which can trigger modification of the current system’s state. Events are time-related: sequences, alternatives, parallelisms, and other synchronization constraints. The interpretation of an event is the record of one of its occurrences along with the triggering of associated processes. For example, “Smith ordered a hammer” is an occurrence of the event “recording an order”, which will be taken in account by the system and execute a process for the actual record of Smith’s order.

3. Semantic contexts

Semantic contexts are associative abstraction mechanisms built over the data structure. Each context gives a particular (and unique) flavour to the association of its components. The interpretation of a context is derived from its components interpretation: each context has a computing rule which produces the context interpretation using the underlying classes populations. A context interpretation only evolves when the state of the system changes.

4. Objects life-cycles

Object life-cycles describe transitions between system states. Their structure is built of pre- and post-conditions expressed on objects states. The interpretation of a transition is its ability (yes or no) to transform a state of the system into another one, i.e. the validity of its pre-condition and the elaboration of resulting post-conditions.
5. **Integrity constraints**

An integrity constraint is a predicate expressed on objects of one system state. The interpretation of a constraint is the truth value of its predicate on a given state. Valid states of an invariant are those for which the predicate evaluates to true.

6. **Periods**

A period is a slice of time during which the whole application field is directed by a unique set of invariant and a unique set of events. The interpretation of a period is defined by an initial instant and a final instant.

Some of these spaces are totally independent (e.g. data and events) whilst some are build using elements of other spaces. Dependencies between spaces are shown in Figure 1. However these dependencies do not induce any constraint on the order in which things must be specified. This order is rather given by the methodology used by the designer.

3  **Models of these six spaces**

3.1  **Data**

A data structure is built with classes, roles, ISA-links, and a set of generic primitives (Create, Delete, Modify, Eval)[10].

- A class is defined by its name and by the applicable primitives. The interpretation of a class is a set of objects represented by their respective object identifiers (oid’s).

- A role is defined by its name, its origin class, and its termination class. It also possesses a minimum and maximum cardinality. The interpretation of a role is a function from the origin class’ interpretation to sets of elements of the termination class’ interpretation. The size of these sets is restricted by the minimum and the maximum roles cardinality. When the minimum and maximum cardinalities of a role are set to one its interpretation is an ordinary single-valued function.

- An ISA-link binds two classes, one being the subclass, the other the superclass, and it is controlled by a predicate. The set of all ISA-links cannot contain
any circuit. An ISA-link imposes the following constraint on the interpretations of its classes: the interpretation of the subclass is exactly the set of all identifiers of the superclass’ interpretation for which the predicate evaluates to true.

At first glance, there isn’t any value in this model. However, objects of some particular classes are self identifying and can thus be considered as atomic values. For instance, the identifier of an object in class `Integer` is the integer number itself. The same applies to predefined classes `Real`, `String`, and `Time`.

The Eval primitive applied to an object identifier o and a role R yields the set of objects of the termination class to which o is mapped by R. Other class primitives are intended to modify the current interpretation of a data structure by adding an identifier to the interpretation of a class (Create), removing an identifier (Delete) or changing a role’s interpretation (Modify). Figure 3 gives a data structure, along with a sequence of primitives and the resulting interpretation.

![Figure 2. Structure of the data space](image1)

![Figure 3. A data structure and its interpretation](image2)
3.2 Events

The event structure is defined by a set of events and a set of associated conditions. A condition is defined by a name. An event is defined by a name and by its pre- and post-conditions.

![Figure 4. Structure of the events space](image)

This structure is defined by successive refinements which preserve some desired properties (like liveliness for example). Refinements primitives express chronological relationships and synchronization constraints over events[12]:

- sequence of events
- alternative between mutually exclusive events
- parallelism of mutually independent events
- simultaneity of events
- precedence of an event on another ($e_1$ before $e_2$)

Moreover, the event structure is equivalent to a Petri net [17] where each event stands for a transition and each condition stands for a place. The interpretation of a condition is the marking of a place, a marked place correspond to a true condition. The interpretation of an event occurrence is the firing of the corresponding transition in the Petri net.

The conservation of dynamic properties has been studied for each primitive so that properties of the resulting event structure can be directly derived from the sequence of used refinements. A specification algorithm is provided to the designer as conceptual framework [8][11].

**Example 1**

Event `pay` is refined by alternatives `pay_cash`, `pay_by_check`, or `by_invoice` and event `by_invoice` is refined by the sequence `send_invoice` then `pay_amount` (see Figure 5).

![Figure 5. A refinements sequence](image)
3.3 Semantic Contexts

A semantic context [8][9][10] is a directed graph built over the data structure. Each node $X_i$ is defined by a name $\text{Name}(X_i)$ and a class $\text{Class}(X_i)$, each directed edge binds two context nodes $X_i$ and $X_j$ and represents one of the roles joining both nodes’ classes $\text{Class}(X_i)$ and $\text{Class}(X_j)$.

A contexts structure (or hierarchy) is a set of semantic contexts ordered by a partial order relation ($\leq$) called the subcontext relation.

Each semantic context has primitives which use the underlying classes primitives in their definition. Figure 6 gives relationships between contexts and data structure concepts.

The interpretation of a semantic context is given by its connection function. Let $k$ be a context and $X = \{X_1, \ldots, X_n\}$ be a subset of its nodes. The connection function maps $X$ and the current interpretation of the data structure to a set of tuples $k[X]$, called the connection of $k$ on $X$. Elements of $k[X]$ are tuples of the form $[X_1: id_1, \ldots, X_n: id_n]$ where each $id_i$ is an element of the interpretation of $\text{Class}(X_i)$. Each tuple of $k[X]$ is thus an association between objects of classes corresponding to $X$’s nodes.

Since there are many ways to define a connection function we state here some condition that a function must satisfy in order to qualify as a connection function:

1. If $X$ is a singleton $\{X_i\}$ then $k[X]$ must be equal to the current interpretation of $\text{Class}(X_i)$,
2. If $X \subseteq Y$ then $\pi_Xk[Y] \subseteq k[X]$, where $\pi_X$ denotes the projection on $X$ of the tuples in $k[Y]$,
3. $k[X]$ depends only on the structure of $k$ and its subcontexts,
4. If $k_i$ is a subcontext of $k$ then $k_i[X] \subseteq k[X]$ for all $X$ in $\text{Nodes}(k_i) \cap \text{Nodes}(k)$.

The last condition implies that a context hierarchy goes from the most general semantics (represented by the maximal contexts) down to the most specialized (represented by minimal contexts).
For example, the connection function of the F2 object-oriented database model [10] is defined as follows:

1. Let \( k(X) \) be the set of all tuples \( [X_1: id_1, \ldots, X_n: id_n] \) such that
   a) \( id_i \) is in the interpretation of \( Class(X_i) \), \( (i = 1, n) \) and
   b) for each pair \( (id_i, id_j) \) and for each path between \( Class(X_i) \) and \( Class(X_j) \) in the data structure, there is a corresponding path between \( id_i \) and \( id_j \) in the interpretation of the data structure.

   If \( X \) is not a subset of \( Nodes(k) \), \( k(X) \) is the empty set.

2. The connection \( k[X] \) is then defined as the union \( \bigcup_{k_i \leq k} k_i(X) \).

Context primitives are:

- \( k.Connect(X) \): evaluates the connection \( k[X] \)
- \( k.Insert(X, x) \): modifies the current interpretation to make tuple \( x \) appear in \( k[X] \).
- \( k.Suppress(X, x) \): modifies the current interpretation to make tuple \( x \) disappear from \( k[X] \).

   Note that if \( x \) is a 1-tuple \( [X_i: id_i] \), \( k.Insert(x) \) simply comes down to the class primitive \( Class(X_i).Create(id_i) \), and similarly for the Suppress and Delete primitives.

**Example 2** Figure 7 shows two contexts: \( Received_Papers \) and \( Papers_Presentation \) defined on the database structure of Figure 3 (each node is labeled by its name and its class). In context \( Received_Papers \) the connection \( Received_Papers[Pa Aut] \) associate papers with their authors while \( Papers_Presentation[AP Aut] \) associate each paper with the person who will present it.

![Figure 7. Two semantic contexts over a data structure](image)

### 3.4 Objects life-cycles

The object life-cycles structure is derived from the event and context structures. Each condition gives rise to a context node and a dynamic state; each event gives rise to a
A process is defined by its name, a context, a set of input connections, a set of output connections, and a set of context primitives acting on these connections. Each context node which appears in an input connection must appear in some output connection.

A dynamic state on objects of a class C is defined by a multi-valued role \texttt{dyn\_state} from C to a class C\texttt{\_dyn\_state} whose objects represent all the possible dynamic states for objects of C. The interpretation of a dynamic state is given by the interpretation of the corresponding \texttt{dyn\_state} role, this gives for each object of C the set of its dynamic states (an object may simultaneously be in several different dynamic states).

In a context dynamic states are defined over connection tuples. The dynamic state of a tuple \([X_1: id_1, \ldots, X_n: id_n]\) is the tuple \([X_1: \text{dyn\_state(id}_1), \ldots, X_n: \text{dyn\_state(id}_n)\]).

The interpretation of a process may be decomposed as follows:
1. Compute the input connections in the process’ context
2. Restrict the connections to tuples whose dynamic states correspond to the input conditions. These tuples are those on which the process is allowed to operate.
3. Select one or more tuples from each restricted connection. These are the tuples which will be actually processed.
4. Do all the internal processing
5. Update the dynamic states of the processed objects and connect or disconnect them so as to put them in states corresponding to the output conditions.

\textbf{Example 3} Creating an association

The process \texttt{affect\_paper\_to\_session} acts in context \texttt{Papers\_Presentation}, it requires two atomic connections, corresponding to individual objects in the desired state. After its execution these two objects must appear in a tuple of the connection \texttt{Papers\_Presentation[AP S]} with state \([AP: <\text{affected}>, S: <\text{defined}>]\).
Since an object may be involved in several different activities of the information system, its dyn\_state role, which is multi-valued, gives its current position in each one of its life-cycles.

### 3.5 Integrity constraints

An integrity constraint is defined by its name, its context, and a predicate on context connections. Integrity constraints usually express data dependencies (existential, functional, inclusion, etc.).

To each integrity constraint is associated a validation automaton which is intended to link the truth value of the constraint’s predicate to the specified processes. The simplest validation automaton is shown in Figure 10.

![Validation Automaton](image)

**Figure 9. A process with two input and one output connection**

**Figure 10. Basic validation automaton**

Each automaton’s transition determines an effect that a process may have on the validity of the constraint. Each automaton’s place represent a validity state for the constraint. The automata may be further refined in order to distinguish between different validity states of the constraint. For instance, one may explain the different reasons for the invalidity of the constraint and show how to come back to a valid state.
Every process is associated to at least one kind of effect for each constraint which represent all the possible validity states that may happen after the process’ execution. If a process is associated to more than one such state it must be refined in several alternative (sub)processes. This will ensure that the objects life-cycle specification yields deterministic information about the validity of the integrity constraints.

**Example 4** Consider the following integrity constraint:

C1: At least \( \min \) and at most \( \max \) papers may be assigned to a session.

with the following refined validation automaton:

![Validation automaton for constraint C1](image)

The process `affect_paper_to_session` of Example 3 may either leave C1 in its current validity state or change from `invalid_under_min` to valid or from valid to `invalid_over_max`. It is thus associated to five transitions of the validation automaton. So it must be refined into five alternatives, namely: `affect_paper_under_min`, `affect_paper_between_min_max`, `affect_paper_over_max` (these ones leave C1 in its current validity state), `affect_paper_min` (make C1 valid), and `affect_paper_max+1` (make C1 `invalid_over_max`).

The interpretation of an integrity constraint is its validity state in the validation automaton. This state must be consistent with the evaluation of the constraint predicate on the current data interpretation, however if processes are refined as indicated above there is no need to evaluate the constraint predicate after each process execution. The firing of the transition associated to the process in the validation automaton gives the new validity state of the constraint.

### 3.6 Periods

A periods structure is made of a set of periods and a set of instants. A period is defined by a name, a set of integrity constraints, and a set of events. An instant is defined by a name. Each period is associated to its starting and ending instants. This structure determines a Petri Net whose places correspond to periods and whose transitions correspond to instants. The interpretation of a periods structure is the assignment of the value “active” or “inactive” to each period. The set of active periods determines the marked places of the periods’ Petri Net. When an instant occurs the corresponding Pet-
ri Net’s transition is fired and the periods’ interpretation is modified according to the new marked places.

We say that data and constraints interpretations are consistent with the periods interpretation if all the constraints belonging to an active period are in a valid state. The events interpretation is said to be consistent with the periods interpretation if the fired transitions of the events’ Petri Net belong to at least one active period.

**Example 5** During the program preparation period all the events `affect_paper_...` of Example 4 are allowed. These events act on the C1 constraint validity, according to its validation automaton. One can see that it is possible to reach a state where C1 is valid and thus to start period Registration.

---

**4 Interactions between the spaces**

**4.1 Data - contexts**

Contexts allow to specify different semantics for the association among a set of classes (corresponding to different logical access paths). The information contents of a context may only be accessed and manipulated through context primitives that act on nodes connections. This isolates the data structure, by hiding the roles from the objects lifecycle, constraints, and periods structures. It is thus possible to change the data structure independently of these three structures, as long as each context is adapted to keep its semantics.

The constraints put on the connection functions assure that contexts faithfully represent the information born by the data interpretation.
4.2 Data - events

These spaces are independent, so this model does not impose a “data-driven” or an “event/process-driven” methodology.

4.3 Contexts - events - objects life-cycles

The interconnection of events and contexts define the objects life-cycles. This interconnection is the part of the object life-cycles that is not deducible from the two previous spaces.

In this space one can pose questions about the reachability of certain data states or about which sequence of processes to execute to attain a desired state.

The dynamic states introduced in the objects life-cycles cannot be reduced to the verification of some integrity constraint. That is, to know whether an object is in a given dynamic state it is not possible to evaluate some predicate on the current data interpretation. This comes from the fact that dynamic states depend on the history of an object and not on its current value.

The definition of object life-cycles allows to check the completeness of data and event structures, since each process must act upon some data and each class must be acted upon (at least by an object creation process).

4.4 Contexts - integrity constraints

The range of an integrity constraint is the context used in the definition of its predicate. This determines the set of classes in which updates may cause the validity state of the constraint to change. In particular, if a process acts in a context that is disjoint from the constraint’s context, one may be sure that it has no effect on the constraint’s validity state.

4.5 Events - integrity constraints

Event occurrences determine integrity constraint validity states through the composition of the events’ Petri Net with all the constraint validation Petri Nets. This makes possible to examine the effect of an events’ sequence on constraints validity.

Thus it is possible to reduce integrity constraint checking to the observation of a Petri Net, without any access to data. The price to pay for that is the need to completely refine each event with respect to each constraint, as indicated in Section 3.5.

4.6 Events - integrity constraints - periods

This interaction defines the global behavior of the system. The event structure together with the periods structure determine, for each period, all the possible transformations (all the trajectories) of the system’s state. Integrity constraint select the transformations that leave the system in a state where all the period’s constraints are verified.
The transition from a period to another one may only happen as both constraints sets are simultaneously being in valid states. This imposes that two mutually incompatible constraints cannot belong to successive periods.

The composition of the objects life-cycle Petri Net with the constraints validation PN, restricted to the events of a period, gives a mean to test if there exists an events’ sequence leading to a state in which the transition to another period is possible.

5 How different models cover these spaces

We shall analyze hereafter the relational and object-oriented models within the framework of the six design spaces we have introduced. Note that this analysis is limited to these spaces and hence do not reflect all the features of these models.

5.1 The relational model

There are several flavors of relational models, the one we shall consider is the model implemented in well known dbms systems such as Oracle, DB2, etc. This includes the data model, query systems and programming languages with embedded database languages.

The relational model is considered as the first model that implemented physical independence between the logical structure of data (the relational schema) and its physical storage structure. Thus the space of data is covered by the model.

We can consider that the notion of view (a derived relation defined by a query expression) partially covers the space of contexts if we identify attributes with context nodes and relations with context edges. Views provide a user interface to data that isolate her or him from the relational schema. However, the function given by the projection of a view’s query do not form an acceptable connection function: views defined by join operations do not satisfy condition 1 and views defined by outer-joins depend on the computation order. In addition, update through views is limited to very simple cases. Thus views are more query macros than semantic contexts.

Integrity constraints predicates may usually be specified using the database query language. The definition of triggers is the only way to integrate integrity constraints into processes but it is limited to elementary processes: create, modify, and delete. This is in no way a specification mechanism that helps in answering questions about the reachability of system states.

The synchronization of events is buried in the database access programs making it impossible to verify the system’s behavior without actually executing the processes. Dynamic states must be directly implemented as attributes in the data structure. The notion of period is absent from the model, in other words, everything happens in the same period.

These considerations show that it is impossible to study global system behavior in information systems based on current relational dbms.
5.2 Object-oriented models

Object-oriented systems have achieved physical independence between data and their representations as well as implementation independence between the specification of a class and the implementation of its methods. In addition, the identity (and the existence) of an object has been made independent of its value.

Events in an object system can be interpreted as messages sent by processes to objects. From a structural point of view, the set of messages accepted by an object is entirely defined by its class interface. But there is no way to know which messages it emits, without reading the implementation of its methods. This “black box” model prevents any knowledge about events synchronization.

Other global specification aspects are missing in object models:

1. Integrity constraints may not be declared, this implies that each constraint must be checked in an ad-hoc way when writing methods’ implementations.
2. There is no general semantic context construct. Thus, abstractions build by connecting different objects must be represented as classes with methods that find connected objects, create connections, and delete connections. As for integrity constraints this leads to put a part of the designer’s knowledge in what should be hidden, namely the methods implementation.

Finally, in object-oriented systems data and event spaces are not independent since the concept of method is subordinate to the concept of class. This need to attach each method to a class leads to a kind of data-driven design.

The work by Kappel and Schreff [15][19], which is an extension of the model defined in [8], is a proposal to integrate events specification and object life cycles in object-oriented design. Their behavior model includes dynamic states and processes (called activities) as well as their refinements. A local object life cycle is specified for every class of the data model; the interaction between objects is then defined by adding pre and post states, taken from other life cycles, to the activities. Thus the global behavior of the system is only known after this final step and is not obtained in a top-down manner as in our approach.

5.3 Logical and behavioral independence

While physical independence is a key concept of the relational model and implementation independence is a key concept of the object-oriented model, our model aims at providing logical and behavioral independence. The concept of dynamic state allows to specify system’s behavior independently from data and process interpretation and the concept of constraint validation state allows to specify system’s integrity without having to evaluate each constraint predicate on each database state.

This behavioral independence makes it possible to answer questions about dynamic aspects of the system by considering only its specifications, i.e. no simulation on test data is required. Thus, conceptual tools to predict system’s global behavior are provided to the designer. In other models, through lack of this global knowledge, the designer has to
— Implement local checking of the constraints, e.g. by evaluating constraints predicates after each action. The lack of period concept even leads to too strong restrictions on acceptable system’s states, because all the constraints must be valid at every instant.

— Accept to be unaware of the reachability of desired states. He can just hope that processes are sufficient and constraints not too restrictive since, with his local view of the system, he is unable to find a consistent sequence of processes that lead to the desired state.

— Specify for each process the logical access paths to use within the data structure and to modify them when a transformation of the data structure occurs.

Logical and behavioral independence have a strong impact on information system’s evolution and maintenance. They limit the consequences on other spaces of a structural change in one space. They allow an incremental development by successive extensions of the different specifications. The global effect of a local specification change can be studied.

In [6] we show that the local behavior of objects and the events specification can be first independently defined and then formally connected to form the global system life cycle specification.

6 Conclusion

The model we have presented here provides an extended descriptive power to the information system designer. In particular, the following aspects can be expressed:

- the semantics of association between objects (semantic contexts);
- temporal dependencies between events (synchronization);
- object state transformations (object life-cycle);
- how processes change the validation state of integrity constraints;
- varying operating conditions (periods).

This conceptual model has also been formally defined and several tools, supporting different design spaces, have been implemented:

- The MTG/r system [11] allows to specify integrity constraints, periods, events on a relational data model. These specifications can then be compiled and executed.

- The F1 dbms implements the Ecrins semantic data model [13],[14] which is accessed exclusively through semantic contexts. A graphical user interface based on semantic contexts has been developed[5].

- The Farandole system integrates the object oriented database model F2 and its semantic contexts [10] into the MTG system. This prototype is used to experiment and validate the design process based on our spaces.
These implementations show that new concepts described here are independent from the data storage system (as a consequence from logical independence). For example, semantic contexts have been formally defined and implemented on relational, semantic and object oriented database systems [9]. In the Farandole system the object life-cycle specification is automatically translated into a set of methods for the F2 object-oriented database.

From a methodological point of view one can remark that the same phenomenon can be equivalently specified in different spaces. For instance, the phenomenon “only accepted papers are presented in conference sessions” can be represented in the following spaces:

- In the data space, by a subclass Accepted_Paper of class Paper, with predicate “status = accepted” and a role from class Session to Accepted_Paper.

- In the constraints space, by a constraint with predicate “for all paper p associated to a session, status(p) = accepted”.

- In the object life-cycles space, by a process Affect_Paper_To_Session which has a pre-condition corresponding to objects of class Paper which are in the dynamic state Accepted.

Since this situation is not forbidden and since no particular priority exists between spaces this induces redundancy in the system’s specification. This redundancy can then be used to check the consistency of specifications. However one should not deduce that one of the spaces is entirely determined by the others, there exists counter-examples showing that each space is conceptually irreducible.

References


