Towards a new infrastructure supporting interoperability of information systems in development: the Information System upon Information Systems

Thang LE DINH

University of Geneva, CUI, 24 General Dufour,
CH-1211 Geneva 4, Switzerland

Thang.LeDinh@cui.unige.ch

Abstract. This paper addresses the issue of interoperability of information systems (IS) in development. Accordingly, an Information System upon Information Systems (ISIS) is proposed as a new infrastructure to manage and coordinate information resources used in IS development process. The proposed approach considers that conceptual specifications of information systems are the fundamental constituents of information resources; and therefore, the first challenge is to manage and coordinate conceptual specifications. Correspondingly, the conceptual framework for building and managing the ISIS is presented, including how to identify, represent, manage and coordinate conceptual specifications.

1. Introduction

Nowadays, information systems (IS) have performed increasingly important roles in both business and governmental sectors. However, after a long period of IS development, most organizations have supported by various heterogeneous applications, software, solutions as well as development tools, which are not designed to coordinate with the others.

As a matter of fact, there is a tendency to require more interoperability of systems and tools used in IS development process. This requirement contains numerous potential advantages, but also leads to certain challenges in the capacity to manage effectively such environments.

Unfortunately, providing interoperability of systems and tools at the Informatics level is not sufficient because systems and tools weren't designed to interoperate at this level [1]. For this reason, our research focuses on providing interoperability at the Information level, which represents the semantic content of information, independent with technologies and choices of implementation.

The objective of this paper is to propose a typical information system that supports interoperability of information systems in development. This information system is called the *Information System upon Information Systems* (ISIS) [4]. The ISIS, which

supports interoperability of information systems in development, will participate in the IS infrastructure hierarchy of each enterprise as a new infrastructure and will coexist with other infrastructures.

On the other hand, we also argue that conceptual specifications are bound to play an important role in the deliverables of the IS development process.

Indeed, specifications have been considered as important information resources that might cover long-term vision of IS. However, once information systems operated, these resources often felt into oblivion. There weren't so much effort to keep these information resources up-to-date.

In our approach, specifications, especially conceptual specifications, are used to capture the knowledge about information systems and used to support interoperability of information systems at the Information level.

Furthermore, most technical specifications are the implementations of conceptual specifications. Therefore, once the interoperability of conceptual specifications is settled at the Information level, the interoperability of the technical specifications and business data at the Informatics level will be solved accordingly.

Consequently, this paper continues with the framework for building and managing the ISIS, including how to identify, represent, manage and coordinate conceptual specifications of information systems.

The remaining of this paper is proposed as the followings: Section 2 presents how to identify different categories of conceptual specifications. Section 3 discusses about the representation of conceptual specifications. Section 4 concerns with the management of conceptual specifications. Section 5 describes how to coordinate specifications within and between development projects. Section 6 comes to end with conclusion and future works.

2. Identification of conceptual specifications

This paper uses the *M7 method* [2, 3, 4] as the tool for representing the framework. The M7 method proposes several models to represent different aspects of an IS such as the Static, Dynamic, and Integrity rule aspects.

The Static aspect represents the structure of information; the Dynamic aspect represents the transformation of information; and the Integrity aspect represents the coherence of information.

2.1 Conceptual specifications of the Static aspect

Conceptual specifications of the Static aspect describe what type of information exists, their structures as well as their interrelationships. There are the categories of conceptual specifications of the Static aspect such as Atomic-class, Tuple-class, Hyperclass, Attribute, Key, and Sub-hyperclass.

An object type and a set of objects of this type define a class. There are three kinds of classes: Atomic-class, Tuple-class and Hyperclass:

- An **atomic-class** is defined as a primitive class, which is indecomposable. Objects of an atomic-class have a particular characteristic: their identifier is also their value.
- A **tuple-class** contains objects having the same structure and the same behaviour. A structure of a tuple-class is characterized by a set of attributes. The behaviour of a tuple-class is represented by a set of methods.
- A **hyperclass** is a subset of classes that all connected by navigation links to a *key class* without ambiguity [5]. We can work on a hyperclass as a tuple-class. Consequently, a tuple-class is a specialisation of a hyperclass.

The interrelationships between the classes' concepts lead to other concepts such as Attribute, Key and Sub-hyperclass:

- An **attribute** of a hyperclass is a function that corresponds to every object of this hyperclass to a set of objects of the other class.
- A **key** of a hyperclass is defined by a set of special attributes can be used to distinguish one object from other objects in the same hyperclass.
- A hyperclass can define its **sub-hyperclasses**. The interpretation of a sub-hyperclass is exactly the set of all identifiers of the interpretation of its super-hyperclass for which the specialisation condition evaluates to be "true".

2.2 Conceptual specifications of the Dynamic aspect

In the M7 method, there are two levels of behaviour: *Local behavior* defined as the behaviour of objects of a hyperclass, and *Global behavior* defined as the behavior of the IS or a part of IS. The categories of conceptual specifications of the Dynamic aspect are Dynamic state, Method, Event, and Process.

The local behaviour is represented by the concepts of *Dynamic State* and *Method:*

- **Dynamic states** of an object are modes or situations during which certain methods are "enabled" and other methods are "disabled".
- A **method** of a hyperclass is used to transit between the dynamic states of the objects of this hyperclass. In a clear manner, a method transfers a set of dynamic states to another set of dynamic states of a hyperclass.

On the other hand, the global behaviour is represented by the concepts of *Event* and *Process*:

- **Event** is remarkable phenomena outside of the information system that may provoke a change of its dynamic states. In fact, the event structure helps to define the interface of the IS with its environments.
- A **process** is a feedback of the IS to the occurrence of an event. In fact, a process performs a transformation of a set of dynamic states of the IS.

2.3 Conceptual specifications of the Integrity rules aspect

The categories of conceptual specifications of the Integrity rule aspect are Integrity rule, Scope, Primitive and Risk.

In fact, the Integrity rule aspect includes the concepts such as *Integrity Rule* and *Primitive* as well as their interrelationships such as *Scope* and *Risk* [6]:

- **Integrity rules** (IR) of an IS often represent the business rules of an organization. An IR actually is a logical condition defined over tuple-classes that can be specified formally and verified by processes or methods.
- **Scopes** of an IR represent the context of an IR including a set of tuple-classes on which the IR has been defined.
- A **primitive** is a basic operation on a tuple-class such as such *Create*, *Update* and *Delete*. The execution of a primitive may violate the validation of an IR.
- **Risks** are the possibilities of suffering the incoherence of information. In principal, a risk is defined on a scope and a primitive. In particular, especially in the case of the *Update* primitive, it is indispensable to specify the related attributes of a risk.

3. Representation of conceptual specifications

This section concerns with how to represent the structure, the behavior and the coherence of conceptual specifications.

3.1 Structure of conceptual specifications

In the ISIS, a conceptual specification is represented by an object. Each category of conceptual specifications is represented by a hyperclass of the ISIS. Hyperclasses of the ISIS are called ***generic hyperclasses.***

Table 1 proposes the generic hyperclasses.

Table 1: Generic hyperclasses.

Aspect	*Generic hyper-class*	*Key class*	*Constituent classes*
Static aspect	Atomic-class	Atomic-class	Category
	Hyperclass	Hyperclass	Category, Tuple-class, Sub-hyperclass, Attribute, and Key
	Attribute	Attribute	Category, and Hyperclass
	Key	Key	Category, Hyperclass, and Attribute
Dynamic aspect	Dynamic state	Dynamic state	Sub-Hyperclass, and Hyperclass
	Method	Method	Dynamic state, Sub-hyperclass, and Hyperclass
	Event	Event	
	Process	Process	Event, Hyperclass, and Method
Integrity rule aspect	Integrity rule	Integrity rule	Integrity rule, Scope, and Risk
	Scope	Scope	Integrity rule, Tuple-class, and Hyperclass
	Risk	Risk	Scope, Primitive and Attribute

Notes: *Category* tuple-class is the generalization of *Atomic-class* tuple-class and *Hyperclass* tuple-class. An object of *Category* tuple-class can be an object of *Atomic-class* tuple-class or an object of *Hyperclass* tuple-class.

For illustrating, the next example presents the *Hyperclass* generic hyperclasses.

Example: Structure of *Hyperclass* specifications.

Hyperclass hyperclass is the generic hyperclass of the ISIS that represents the specifications of hyperclasses of information systems.

Hyperclass hyperclass is defined over its key class: *Hyperclass* tuple-class.

Objects of Hyperclass tuple-class are objects of *Category* tuple-class. In the same manner, objects of *Sub-hyperclass* and *Tuple-class* tuple-classes are also objects of Hyperclass tuple-class and therefore objects of *Category* tuple-class.

From an object of *Hyperclass*, one can navigate to an object of *Sub-Hyperclass* tuple-class (using its *sub-hyperclass-of* attribute), a set of objects of *Attribute* tuple-class (using the *origin-Hyperclass* attribute), and a set of objects of *Key* tuple-class (using the *key-of-Hyperclass* attribute).

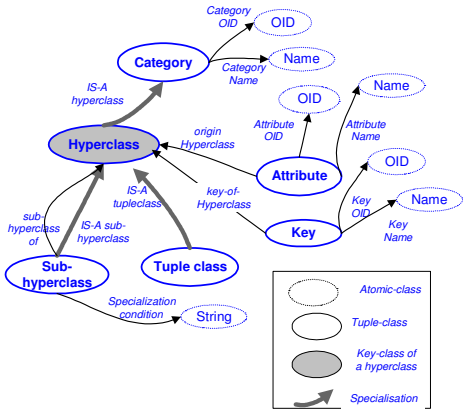

Figure 1: Structure of Hyperclass generic hyperclass.

3.2 Coherence of conceptual specifications

The coherence of conceptual specifications is guaranteed by the integrity rules of the ISIS. In our framework, the integrity rules of the ISIS are called ***generic integrity rules***. Indeed, these integrity rules concerns with the conformity of conceptual specifications.

An object of a generic hyperclass of the ISIS is said: "to be conformed" if it is satisfied all the generic integrity rules. There are two types of generic integrity rules: *validity* and *completeness rules*.

The concept of **validity rule** is actually inherent to the concept of "integrity rule" at the meta-model level. Indeed, there is a set of rules coordinating with the meta-model to control the validity of every object of generic hyperclasses.

When designer modifies an object of a generic hyperclass, this modification may violate the validity rules, which are concerned about this object. If the modification violates one of those rules, the object is brought into the *invalid* state. On the contrary, it is in the *valid* state.

For instance, there is a validity rule related to objects of Hyperclass generic hyperclass such as: "*The dependent constituents of a hyperclass such as its attributes, keys, and sub-hyperclasses must be valid"*.

The concept of **completeness rule** is related to the perception about the organization and the real world that designers have to realize. In fact, the decision of designers about the completeness of a conceptual specification depends on the finish of works and the stability of the constituents of the real world, which are modelled with that specification. Therefore, designers who are responsible for managing completeness rules will decide their completeness status.

For example, "*a hyperclass must have all its attributes*" is a completeness rule. In this case, the designer, who decides whether all the "necessary" attributes of that hyperclass are already specified or not, will specify its completeness status.

3.3 The behaviour of conceptual specifications

The behaviour of conceptual specification is represented by the generic dynamic states of conceptual specifications and the corresponding object life cycles.

A ***generic dynamic state*** is a dynamic state of the ISIS that is common for all the object life cycles of categories of conceptual specifications. In other words, those dynamic states exist in all the object life cycles.

We propose the following dynamic states: *Ready to initialize, Initialized, Valid, Invalid, Completed, Uncompleted, Implemented,* and *Unimplemented.*

Firstly, every object of a generic hyperclass, which represents a conceptual specification, has two dynamic states: **Ready to initialize** before its initiation and **Initialized** after its initiation.

Secondly, when an object is initialized, it is said: "to be conformed" if it is satisfied all the conformity rules, including validity and completeness rules.

Therefore, the next dynamic states are proposed:

- **Valid** / **Invalid** to state that an object is satisfied /dissatisfied all the concerned validity rules;
- **Completed** / **Uncompleted** to state that an object is satisfied /dissatisfied all the concerned completeness rules.

Finally, the dynamic states to indicate that an object of a generic hyperclass is implemented (or not) are also necessary. Consequently, the two generic dynamic states: **Implemented** and **Unimplemented** are also proposed.

In the ISIS approach, the development process of a conceptual specification can be generally represented by a ***generic object life cycle***. A method of the ISIS that may change the generic dynamic states of an object is called a ***generic method.*** Figure 2 presents the generic object life cycle using the Petri-net [7].

An object before its initiation is in the *Ready to initialize* state. After its initiation (using *Initialize()* method), the object is in the *Initialized* state and ready for processing by other methods. For instance, it is ready to be queried by the *Query()* method. When an object is created, it is also in the *Invalid*, *Uncompleted* and *Unimplemented* states.

A set of *Specify()* generic methods can be used to bring an object from the *Invalid* state into the *Valid* state or vice versa. Accordingly, a set of *Create_dependent()* generic methods can be used to bring an object from the *Uncompleted* state into the *Completed* states. On the other hand, the *Delete_dependent()* generic method may bring an object return to the *Uncompleted* state.

An object can be implemented when it is in *Valid* state. When an object is in the *Unimplemented* state, there are two possibilities:

- This object is valid but has not yet completed, however, the responsible person decides to implement it. In that case, an *Uncompleted-implement()* method change its state into *Implemented*;
- This object is *Valid* and *Completed*, and then a *Completed-implement()* method changes its state into *Implemented*.

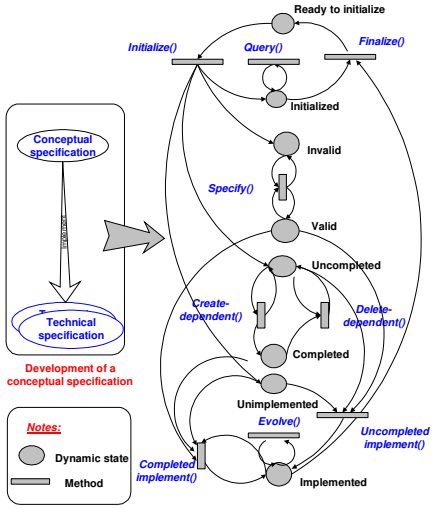

Figure 2: A generic object life cycle.

Moreover, when an object is in the *Implemented* state, there are two possibilities:

- This object may be completed or not yet completed, therefore a *Completed-implement()* method or *Uncompleted-implement()* method can be executed. Those methods do not change its state;
- There is a need for an evolution. In that case, the *Evolve()* method can perform the evolution primitives. This type of methods does not change the state of object.

Finally, an object in the *Initialized* state, probably implemented or not implemented, can be finalized by using the *Finalize()* method. This method brings the object return to the *Ready to initialize* state.

4. Management of conceptual specifications

This section firstly present how the ISIS store conceptual specifications, then continues with the overall architecture of the ISIS.

4.1 Organizational aspect of the ISIS

An excerpt of the meta-model of the organizational aspect of the ISIS presents how the ISIS store and manage the conceptual specifications (Figure 3).

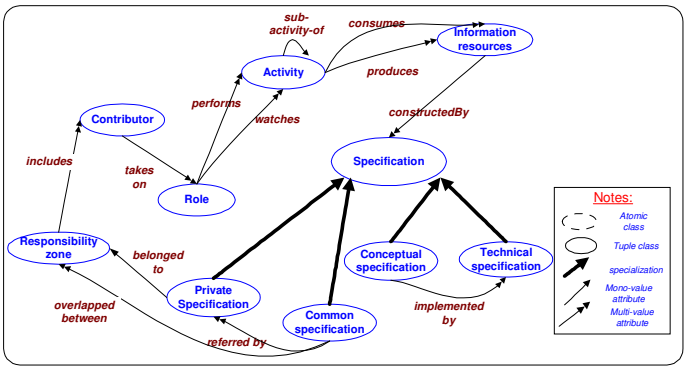

Figure 3: Meta-model of the Organizational aspect of the ISIS.

Concerning the levels of abstraction, there are two types of specifications: *Conceptual specification* and *Technical specifications*. Technical specifications represent the internal design of the information system, turned to achieve reasonable performance on the target platform. A conceptual specification can be implemented in one or several technical specifications.

An *information resource* is a package of specifications that are consumed or produced by the activities of the development process.

An *activity* is defined as a unit of work that may produce or consume certain information resources. Activities can be nested: one activity can expand into several activities at a lower level. An activity can be performed by a set of roles and can be watched by another set of roles.

Roles represent a set of necessary responsibilities, authorities and capabilities to perform the execution of activities or to watch (monitor) the execution of activities performed by the other roles.

A *contributor* is a person that participates in the IS development process. A contributor may take on several roles.

A *responsibility zone* (RZ) is a part of working environment that may carry out an IS development project. A RZ includes a set contributors associated together.

Concerning the interest of RZs, there are two specializations of specification: *Private specification* and *Common specification*. A private specification is belonged to only one RZ. Meanwhile, a common specification is overlapped between several RZs. Consequently, a common specification can be referred by several private specifications.

4.2 Overall architecture of the ISIS

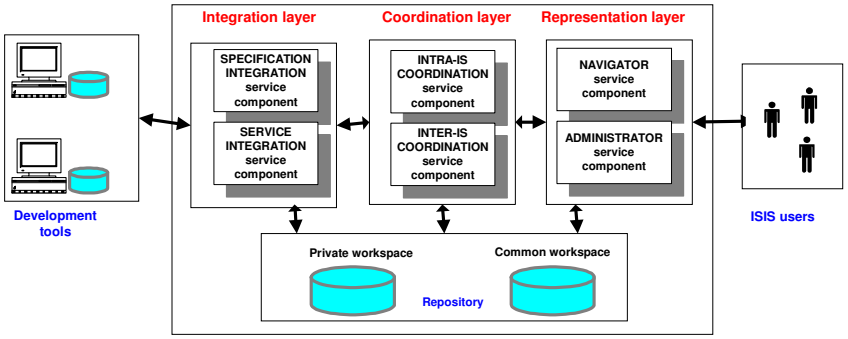

Figure 4: Overall architecture of the ISIS.

Figure 4 introduces the overall architecture of the ISIS. In the ISIS, specifications are stored in the ISIS repository. This repository includes two workspaces: *Private workspace* and *Common workspace*. Private workspace stores private specifications of responsibility zones. Meanwhile, common workspace stores common specifications.

On the other hand, there are three layers that provide facilities to represent, validate, manage, integrate and coordinate specifications such as the Integration, Coordination and Representation layers.

Integration layer provides the facilities to integrate specifications stored in development tools and specifications stored in the ISIS. This layer includes:

- *Specification-integration service component*: providing facilities to integrate specifications stored in tools with specifications stored in the ISIS;
- *Service-integration service component*: including facilities to integrate services provided by tools and services provided by the ISIS.

Representation layer provides the facilities so that ISIS users, developers and administrators can work with specifications stored in the ISIS repository, including:

- *Navigator service component*: providing the interface to allow the ISIS users to specify and complete the specifications;
- *Administrator service component*: providing facilities to support the ISIS administrator.

Coordination layer provides the facilities to support interoperability of specifications within and between responsibility zones. This layer includes:

- *Intra-IS coordination service component*: supporting interoperability of private specifications of the same RZ;
- *Inter-IS coordination service component:* supporting interoperability of common specifications of different RZs.

5. Coordination of conceptual specifications

In the following, we discuss about various situations of coordination and illustrate how an ISIS may support different situations of coordination. Indeed, there are two general situations of coordination: Intra-IS coordination and Inter-IS coordination.

Intra-IS coordination is the coordination of private conceptual specifications managed by a same responsibility zone. Meanwhile, *Inter-IS coordination* is the coordination of common conceptual specifications shared by different responsibility zones.

5.1 Intra-IS coordination

In the ISIS, the development process of a conceptual specification is represented by an object life cycle of the corresponding generic hyperclass. Therefore, the interdependencies between conceptual specifications can be analyzed based on the interdependencies of dynamic states of those life cycles.

In fact, when a conceptual specification changes its dynamic states, this change may lead to the changes of dynamic states of other conceptual specifications, which has the interrelationship with that conceptual specification.

For instance, "*when a conceptual specification of a hyperclass becomes Valid, it is assured that the all its sub-hyperclasses, attributes, and keys must be in Valid dynamic state*".

The impact as mentioned above can be implemented using coordination rules. In other words, to support the coordination of object life cycles of generic hyperclasses, the ISIS needs to guarantee the coordination rules, which represent the impact of the changes of dynamic states of objects of generic hyperclasses.

For instance, the next table presents the coordination rules related to the Hyperclass generic hyperclass. Those coordination rules concerns with the coordination of dynamic states of objects of *Hyperclass* and its dependent generic hyperclasses (such as *Sub-hyperclass*, *Attribute* and *Key*).

Table 2: Coordination rules concerning the coordination of objects of the Hyperclass generic hyperclasses with objects of other generic hyperclasses.

Rule	***Generic hyperclass***	***Description***
S_Cn#1	Hyperclass	If a hyperclass is in V*alid* state then its sub-hyperclasses, attributes, and keys must be also in *Valid* state.
S_Cn#2	Hyperclass	If a hyperclass is in *Invalid* state then its parent hyperclass is also in *Invalid* state.
S_Cn#3	Hyperclass	If a hyperclass is in *Completed* state then its sub-hyperclasses, attributes, and keys must be in *Completed* state.
S_Cn#4	Hyperclass	If a hyperclass is in *Uncompleted* state then its parent hyperclass is in *Uncompleted* state.

5.2 Inter-IS coordination

Inter-IS coordination concerns with common conceptual specifications, which reflect the interdependences between different responsibility zones. These interdependences can be represented by *overlap situations* and overlap situations are operated by *overlap protocols*.

An **overlap situation** occurs when there is at least one class or one process is common to several RZs. There are three types of overlap situations:

- *Distinct:* there is no common class and no common process between RZs.
- *With borders:* there are common classes, but no common process.
- *With overlaps*: there are common classes, and common processes, which perform operations on those common classes.

An **overlap protocol** is a protocol that allows a RZ to perform its own processes locally and to monitor the processes in other RZs, which can influence its own processes.

In fact, an overlap protocol includes a set of semantics, rules, and formats that conduct the coordination of different RZs. At the time being, we propose the following categories of overlap protocols:

- *Ownership-based overlap protocol* appoints which RZ would play the role of the *owner* for each common object. The owner of an object takes the responsibility for defining, developing and maintaining it. The other RZs may communicate to the owner to obtain information about this object.
- *Service-based overlap protocol* appoints which RZ would play the role of the *provider* for each common object. The provider of an object takes the responsibility for providing services related to this object. This protocol allows other RZs to send a *request* to perform a process to the provider. Normally, the pro-

vider will perform the requested process and return the result to the requested RZ.

- *Watch-based overlap protocol* allows a RZ to monitor the consequences when other RZs performed a process, which is overlapped between them. Indeed, each RZ plays the role of the *co-owner* for each common object.

6. Conclusion

In this paper, we have shown up the important of a new infrastructure based on *the Information System upon Information Systems* (ISIS), which supports interoperability of information systems in development. We also proposed a conceptual framework for building and managing the ISIS based on conceptual specifications.

The contribution of our work is to provide a unique and coherent framework to represent, coordinate and validate conceptual specifications of information systems. The perspective of this work is to provide an effective architecture that would be best suited for the interoperability of existing tools and systems used in IS development based on conceptual specifications.

In future work, we will focus our research on designing and building a tool that supports the development of the ISIS, in particular the modelling phase. This tool will help the IS professionals to adapt and to build their own ISIS conforming to their enterprise. For instance, they can define and represent different categories of specifications conforming to their development methods, and select the overlap protocols conforming to their culture, organization styles, and existing technologies.

References

1. M. Stonebraker, “Integrating Islands of Information”, EAI Journal, Sept 1999.
2. Th. Estier, G. Falquet, J. Guyot, and M. Léonard, “ Six Spaces for Global Information Systems Design ”, Proc. of IFIP Working Conference on The Object-Oriented Approach in Information Systems, Québec, Canada, October 1991
3. Th. Estier, "Intégration des spécifications dans la conception des systèmes d'information, Thèse de doctorat de l'Université de Genève, Faculté des Sciences Economiques et Sociales, 1996.
4. T.Le Dinh, “Information System upon Information System: A conceptual framework“, Ph.D thesis, No 577, Faculté des Sciences Economiques et Sociales, University of Geneva, 2004.
5. Slim Turki, , "Hyperclass: Towards a new kind of independence of the method from the schema", ICEIS, Ciudad Real Spain, 2002.
6. M. Léonard, “Information System Engineering getting out of classical System Engineering”, keynote lecture, 5th ICEIS, Angers – France, April 2003.
7. J.L. Peterson, "Petri net theory and the modeling of systems", Prentice Hall, 1981.