Evolution Models for Information Systems Evolution Steering

Jolita Ralyté(✉) and Michel Léonard

Institute of Information Service Science, University of Geneva, Geneva, Switzerland
{jolita.ralyte,michel.leonard}@unige.ch

Abstract. Sustainability of enterprise Information Systems (ISs) largely depends on the quality of their evolution process and the ability of the IS evolution steering officers to deal with complex IS evolution situations. Inspired by Olivé [1] who promotes conceptual schema-centric IS development, we argue that conceptual models should also be the centre of IS evolution steering. For this purpose we have developed a conceptual framework for IS evolution steering that contains several interrelated models. In this paper we present a part of this framework dedicated to the operationalization of IS evolution – the evolution metamodel. This metamodel is composed of two interrelated views, namely structural and lifecycle, that allow to define respectively the structure of a particular IS evolution and its behaviour at different levels of granularity.

Keywords: Information systems evolution · IS evolution steering · IS evolution structure · IS evolution lifecycle · IS evolution impact

1 Introduction

No matter the type and the size of the organization (public or private, big or small), sustainability of its Information Systems (ISs) is of prime importance to ensure its activity and prosperity. Sustainability of ISs largely depends on the quality of their evolution process and the ability of the officers handling it to deal with complex and uncertain IS evolution situations. There are several factors that make these situations complex, such as: proliferation of ISs in the organization and their overlap, independent evolution of each IS, non-existence of tools supporting IS evolution steering, various IS dimensions to be taken into account, etc. Indeed, during an IS evolution not only its information dimension (the structure, availability and integrity of data) is at stake. IS evolution officers have also to pay attention to its activity dimension (the changes in enterprise business activity supported be the IS), the regulatory dimension (the guarantee of IS compliance with enterprise regulation policies), and the technology dimension (the implementation and integration aspects).

In this context, we claim that there is a need for an informational engineering approach supporting IS evolution steering, allowing to obtain all the necessary

G. Poels et al. (Eds.): PoEM 2017, LNBIP 305, pp. 221–235, 2017.
https://doi.org/10.1007/978-3-319-70241-4_15

information for an IS evolution at hand, to define and plan the evolution and to assess its impact on the organization and it ISs. We found the development of such an approach on conceptual modelling by designing a conceptual framework for IS evolution steering. Some parts of this framework were presented in [2,3]. In this paper we pursue our work and present one of its components – the operationalization of the IS evolution through the metamodel of IS Evolution.

The rest of the paper is organized as follows: in Sect. 2 we overview the context of our work – the conceptual framework that we are developing to support the IS evolution steering. Then, in Sect. 3, we discuss the role and principles of conceptual modelling in handling IS evolution. In Sects. 4 and 5 we present our metamodel formalizing the IS evolution, its structural and lifecycle views, and illustrate their usage in Sect. 6. Section 7 concludes the paper.

2 Context: A Framework for IS Evolution Steering

With our conceptual framework for IS evolution steering we aim to face the following challenges: (1) steering the IS evolution requires a thorough understanding of the underpinning IS domain, (2) the impact of IS evolution is difficult to predict and the simulation could help to take evolution decisions, (3) the complexity of IS evolution is due to the multiple dimensions (i.e. activity, regulation, information, technology) to be taken into account, and (4) the guidance for IS evolution steering is almost non-existent, and therefore needs to be developed.

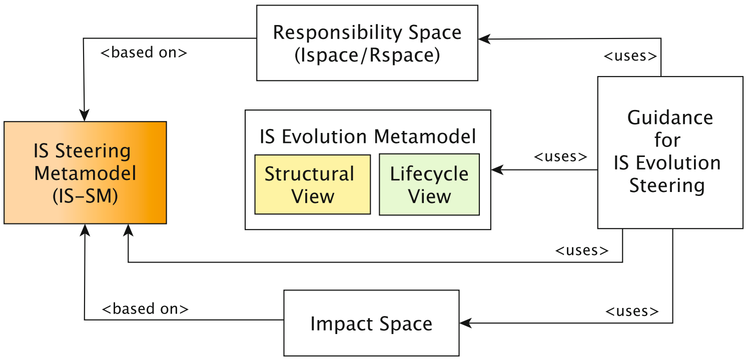

Fig. 1. Conceptual framework for IS evolution steering

As shown in Fig. 1, the framework contains several components each of them taking into account a particular aspect of IS evolution steering and considering the evolution challenges listed above. Let us briefly introduce these components.

The *IS Steering Metamodel* (*IS-SM*) is the main component of the framework having as a role to represent the IS domain of an enterprise. Concretely, it allows to formalize the way the enterprise ISs are implemented (their structure in terms of classes, operations, integrity rules, etc.), the way they support enterprise business and management activities (the definition of enterprise units, activities, positions, business rules, etc.), and how they comply with regulations governing these activities (the definition of regulatory concepts, rules and roles). Although IS-SM is not the main subject of this paper (it was presented in [2,3]), it remains the bedrock of the framework, and is necessary to be presented for better understanding other models and illustrations. IS-SM is also the kernel model for implementing an Informational Steering Information System – ISIS. ISIS is a meta-IS for steering enterprise IS (IS upon ISs according to [4]). While enterprise ISs operate at business level, ISIS performs at the IS steering level. Therefore, we depict IS-SM in Fig. 2 mainly to make this paper self-explanatory, and we invite the reader to look at [5] for further details.

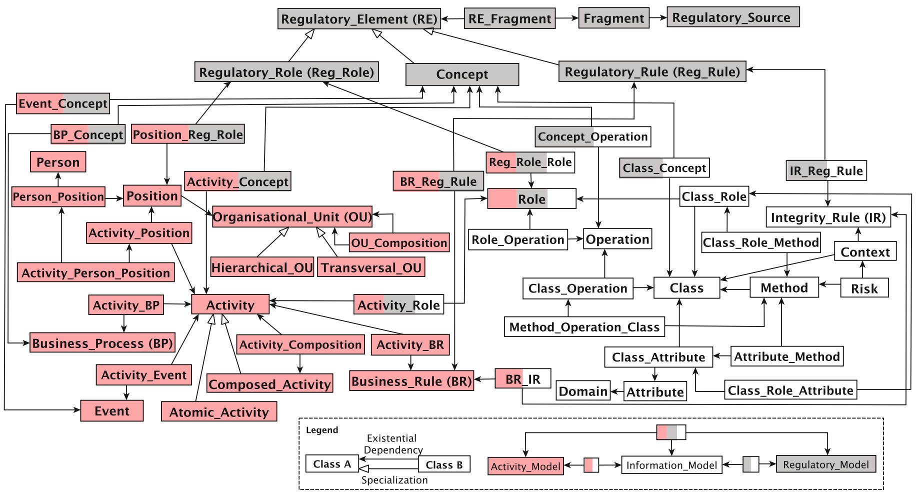

Fig. 2. Simplified version of IS-SM. The right part (in white) shows the information model generic to any IS implementation, the left part (in red) represents enterprise business activity model, the top part (in grey) represents the regulatory model governing enterprise business and IS implementations. The multi-coloured elements represent pivot elements allowing to interconnect the information, activity and regulation models, and so, to capture how ISs support enterprise activities and comply with regulations. (Color figure online)

The role of the *Evolution Metamodel* is to specify IS changes, and to assist the IS steering actor responsible for performing these changes. This metamodel comprises two interrelated views: structural and lifecycle. While the former deals with the extent and complexity of the IS evolution, the later supports its planning and execution. The Evolution Metamodel is the main subject of this paper, and is detailed in the following sections.

The *Impact Space* component provides mechanisms to measure the impact of IS changes on the enterprise IS, on the business activities supported by these

ISs, and on the compliance with regulations governing enterprise activities. The impact model of a particular IS evolution is defined as a part of the IS-SM including the IS-SM elements that are directly or indirectly concerned by this evolution. An IS-SM element is directly concerned by the evolution if its instances undergo modifications, i.e. one or more instances of this element are created, enabled, disabled, modified, or deleted. An IS-SM element is indirectly concerned by the evolution if there is no modification on its instances but they have to be known to make appropriate decisions when executing the evolution.

The *Responsibility Space* (*Ispace/Rspace*) component [3] helps to deal with responsibility issues related to a particular IS evolution. Indeed, each IS change usually concerns one or several IS actors (i.e. IS users) by transforming their information and/or regulation spaces (Ispace/Rspace). An IS actor can see her information/regulation space be reduced (e.g. some information is not accessible anymore) or in the contrary increased (e.g. new information is available, new actions has to be performed, new regulations has to be observed). In both cases the responsibility of the IS actor over these spaces is at stake. The Ispace/Rspace model is defined as a part of IS-SM. It allows for each IS evolution to create subsets of information, extracted from ISIS, that inform the IS steering officer how this evolution affects the responsibility of IS users.

Finally, the *Evolution Steering Method* provides guidelines to use all these aforementioned models when executing an IS evolution.

3 Modelling IS Evolution: Background and Principles

3.1 Background

Most of the approaches dealing with IS and software evolution are based on models and metamodels (e.g. [6–10]). They mainly address the structural aspects of IS evolution (for example, changing a hierarchy of classes, adding a new class) [6], model evolution and transformations [7], and the traceability of changes [9,10]. They aim to support model-driven IS development, the automation of data migration, the evaluation of the impact of metamodel changes on models, the development of forward-, reverse-, and re-engineering techniques, the recording of models history, etc. The importance and impact of model evolution is also studied in [11] where the authors stress that understanding and handling IS evolution requires models, model evolution techniques, metrics to measure model changes and guidelines for taking decisions.

In our work, we also claim that the purpose of conceptual modelling in IS evolution steering is manifold, it includes the understanding, building, deciding and realising the intended IS changes. As per [12], the notion of IS evolution has to be considered as a noun and as a verb. As a noun it refers to the question "what" – the understanding of the IS evolution phenomenon and its properties. While as a verb, it refers to the questions "how" – the theories, languages, activities and tools which are required to evolve a software. Our metamodel for IS evolution steering (see Fig. 1) includes two complementary views, namely structural and lifecycle view, and so serves to cope with complex IS artefacts, usually having multiple views.

Models are also known as a good support for taking decisions. In case of IS evolution, usually, there are several possible ways to realise it, each of them having a different impact on enterprise ISs and even on its activities. Taking a decision without any appropriate support can be difficult and very stressful task. Finally, with a set of models, the realisation of IS evolution is assisted in each evolution step and each IS dimension.

3.2 Principles of IS Evolution

The focus of the IS evolution is to transform a current IS schema (ASIS-IS) into a new one (TOBE-IS), and to transfer ASIS-IS objects into TOBE-IS objects. We use ISIS (see the definition in Sect. 2), whose conceptual schema is represented by IS-SM (Fig. 2), as a support to handle IS evolution. Indeed, ISIS provides a thorough, substantial information on the IS structure and usage, which, combined with other information outside of ISIS, is crucial to decide the IS evolution to pursue. Furthermore, ISIS is the centre of the management and the execution of the IS evolution processes both at the organizational and informatics levels. So, one main principle of IS evolution is always to consider these two interrelated levels: the ISIS and IS levels with their horizontal effects concerning only one level, and their vertical effects concerning both levels. In the following, to make a clear distinction between the IS and ISIS levels, we use the concepts of "class" and "object" at the IS schema level, and "element" and "instance" at the ISIS schema level.

IS evolution is composition of transformation operations, where the most simple ones are called atomic evolution primitives. Obtaining an initial list of atomic evolution primitives for an IS and its ISIS is simple: we have to consider all the elements of the ISIS schema, and, for each of them, all the primitives usually defined over an element: Search, Create, Read, Update, Delete (SCRUD). In the case of IS-SM as ISIS schema, there are 53 elements and so, 265 atomic evolution primitives. Since the aim of the paper is to present the principles of our framework for IS evolution steering, we simplify this situation by considering only the most difficult primitives Create and Delete. Nevertheless, there are still 106 primitives to be considered. The proposed conceptual framework for IS evolution steering is going to help facing this complexity.

4 Structural View of IS Evolution

An IS evolution transforms a part of the ASIS-IS *ISP* into *ISP'*, which is a part of the TOBE-IS, in a way that the TOBE-IS is compliant with:

- the *horizontal perspective*: the instances of the new ISIS and the objects of TOBE-IS validate all the integrity rules defined respectively over ISIS and TOBE-IS;

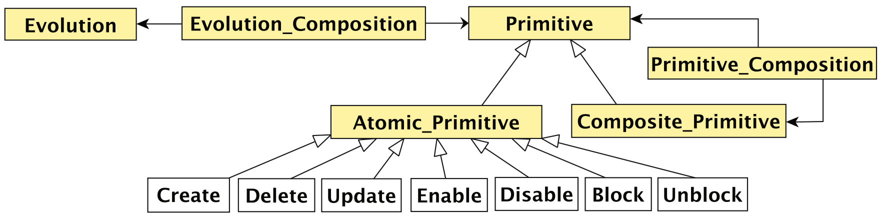

Fig. 3. Structural view of the IS evolution

– the *vertical perspective*: the TOBE-IS objects are compliant with the instances of the new ISIS.

In a generic way, we consider that an overall IS evolution requires to be decomposed into several IS evolutions, and so the role of the structural IS evolution model view (shown in Fig. 3) consists in establishing the schema of each IS evolution as a schema of composition of evolution primitives defined over IS-SM to pursue the undertaken IS evolution.

An evolution primitive represents a kind of elementary particle of an evolution: we cannot split it into several parts without loosing qualities in terms of manageable changes and effects, robustness, smartness and performances, introduced in a following paragraph. The most basic evolution primitives are the atomic evolution primitives: some of them, like Create, Delete and Update, are classic, since the other, Enable, Disable, Block, Unblock are crucial for the evolution process.

4.1 Atomic Evolution Primitives

Since the ISIS schema (i.e. IS-SM) is built only by means of existentially dependencies[1], the starting point of the IS evolution decomposition is very simple – it consists of a list of atomic primitives: *create*, *delete*, *update*, *enable*, *disable*, *block* and *unblock* an instance of any IS-SM element. We apply the same principle at the IS level, so the IS schema steered by ISIS is also built by using only existential dependencies. Moreover, an instance/object is existentially dependent on its element/class.

These atomic primitives determine a set of possible states that any ISIS instance (as well as IS object) could have, namely *created*, *enabled*, *blocked*, *disabled*, and *deleted*. Figure 4 provides the generic life cycle of an instance/object.

Once an instance is created, it must be prepared to be enabled, and so to be used at the IS level. For example, a created class can be enabled, and so to have objects at the IS level, only if its methods validate all the integrity rules whose

[1] A class C2 is existentially dependent on the class C1, if every object o2 of C2 is permanently associated to exactly one object o1 of C1; o2 is said to be existentially dependent on o1. The existential dependency is a transitive relation. One of its particular cases is the specialization.

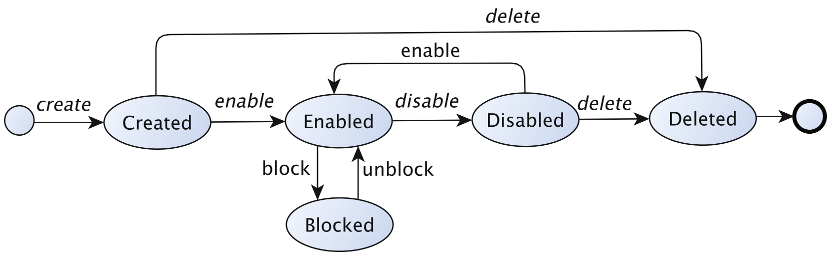

Fig. 4. Lifecycle of an instance of any element from IS-SM

contexts contain it. A created instance can be deleted if it belongs to a stopped evolution.

Enabled instances are disabled by an evolution when they do not play any role in the targeted TOBE-IS. They are not deleted immediately for two reasons: the first one concerns the fact that data, operations, or rules related to them, which were valid before the evolution, still stay consistent for situations where continuity is mandatory, for instance due to contracts. The second one concerns the evolution itself: if it fails, it is necessary to come back to the ASIS-IS, and so, to enable again the disabled instances.

Enabled instances are blocked during a phase of an evolution process when it is necessary to avoid their possible uses at the IS level through objects and/or execution of operations. At the end of this phase they are unblocked (re-enabled). For instance an activity (an instance of the element Activity) can be blocked temporary because of the introduction of a new business rule. Finally when an instance is deleted, it disappears definitively.

4.2 Robust Generic Atomic Evolution Rules

The generic atomic evolution rules must be validated to assure the consistency of the evolution process. Indeed each atomic evolution primitive has effects on other elements than its targeted elements. For example, deleting an integrity rule has effects on the methods of several classes belonging to the context of this integrity rule. Below, we present two kinds of generic atomic evolution rules: the first concerns the horizontal and vertical evolution effects, while the second deals with the dynamic effects.

Evolution Effects Horizontally and Vertically. An evolution primitive is firstly an atomic operation on the ISIS. So, it must verify the integrity rules defined over the IS-SM model to manage the *horizontal effects*. For example, if an instance cl of the element Class is deleted, then all the instances clo_i of the element Class_Operation related with cl must be deleted due to the existential dependency between these two elements (see Fig. 2).

An evolution primitive is also an operation on the IS and has to manage the *vertical effects* of the conformity rules between ISIS instances and IS objects. For example, deleting *cl* induces also deleting all its objects in the IS.

Then, since the evolution operations on IS are executed from ISIS, they validate the integrity rules defined over IS, which are instances of ISIS.

Generic Dynamic Evolution Rules. The generic evolution rules concern the states of the ISIS elements produced by the use of atomic evolution primitives (Fig. 4): created, enabled, blocked, disabled, deleted, and especially the interactions between instances of different elements in different states. They must be observed only at the ISIS level.

Some generic rules concerning the states "created" and "deleted" are derived directly from the existentially dependencies. Considering the element *Einf* depending existentially on the element *Esup*, any instance of *Einf* may be in the state "created" only if its associated instance *esup* of *Esup* is in the state "created", and it must be in the state "deleted" if *esup* is in the state "deleted".

The generic rules concerning the states "blocked" and "disabled" require to consider another relation between the IS-SM elements, called "determined by", defined at the conceptual level and not the instance level. An element *Esecond* is strictly/weakly determined by the element *Efirst* if any instance *esecond* to be exploitable in IS must/can require to be associated to one or several instances *efirst*.

Then there is the following generic dynamic rule: any instance *esecond* must be in the state disabled/blocked/deleted if at least one of its *efirst* is in the state respectively disabled/blocked/deleted.

For instance, the element Operation is *strictly determined by* the element Class, because any operation to be executed at the IS level must be associated to at least one class (see Fig. 2). Then, if an operation is associated to one class in the state disabled/blocked, it also must be in the state disabled/blocked, even if it is also associated to other enabled classes.

The element Integrity_Rule is *weakly determined by* the element Business_Rule because integrity rules are not mandatory associated with a business rule. In the same way, all elements, like Class, associated with the Regulatory_Element are weakly determined by it, because their instances are not mandatory associated to an instance of Regulatory_Element.

Considering the following elements of the IS-SM models (see Fig. 2): Person, Position, Business_Process (BP), Activity, Business_Rule (BR), Role, Operation, Class, Integrity_Rule (IR), Regulatory_Element (RE), here is the list of relations *strictly determined by* (=>): BP => Activity, BR => Activity, Operation => Class, Operation => IR, IR => Class. The list of the relations *weakly determined by* (–>) (in addition to the aforementioned ones with the Regulatory_Element) is: IR –> BR, IR –> RE, Class –> RE, Operation –> RE, Role –> RE, BR –> RE, Activity –> RE, Position –> RE, Event –> RE, BP –> RE.

Robustness. Every aforementioned evolution primitive is *robust* if it manages all its horizontal and vertical effects and respects all the generic dynamic evolution rules. The use of only existential dependencies at the both levels, IS and ISIS, in our approach, facilitates reaching this quality. Nevertheless, at the IS level, such an approach requires that the whole IS schema (including static, dynamic and integrity rule perspectives) must be easily evolvable, and the IT system supporting the IS (e.g. a database management system) must provide an efficient set of evolution primitives [13].

4.3 Composite Evolution Primitives

The composite primitives are built by a composition of the atomic ones (Fig. 3). They are necessary to consider IS evolution at the management level [3], but also for informational and implementation purposes. For instance, replacing an integrity rule by a new one can be considered logically equivalent to delete it and then to create the new one. But this logic is not pertinent if we consider the managerial, IS exploitation and implementation perspectives. It is much more efficient to build a composite evolution primitive "replace" built from the atomic primitives "create" and "delete".

A composite evolution primitive is robust, if it manages all its horizontal and vertical effects and respects all the generic dynamic evolution rules.

4.4 Managerial Effects

The managerial effects consider the effects of the IS evolution at the human level, and so concern the IS-SM elements Role, Activity, Position and Person. The evolution steering officers have to be able to assess whether the proposed evolution has a harmful effect on organization's activities or not, and to decide to continue or not this evolution. The evolution primitives are *smart* if they alert these levels by establishing a report of changes to all the concerned roles, activities, positions, and persons. To do that, they will use the responsibility space (Fig. 1) with its two sub-spaces: its informational space (Ispace) and its regulatory space (Rspace). This part was presented in [3]. Below in the paper, all primitives are smart.

5 Lifecycle View of IS Evolution

Evolution of an information system is generally a delicate process for an enterprise for several reasons. First, it cannot be realized by stopping the whole IS because all the activities supported by IS should be stopped and this situation is unthinkable in most cases. Second, it has impacts, especially on actors and on the organization of activities. It can even induce the need for reorganizing the enterprise. Third, it takes time and often requires to set up a process of adaptation to the changes for all concerned actors to enable them to perform their activities. Moreover, it concerns a large informational space of IS-SM and requires to be decomposed into partial evolutions called sub-evolutions. So, it requires a coordination model to synchronize all processes of these sub-evolutions as well as the process of the main evolution. Furthermore, it is a long process, with an important number of actors who work inside the evolution process or whose activities are changed by the evolution. Finally, most evolutions of ASIS-IS into TOBE-IS are generally nearly irreversible, because it is practically impossible to transform back TOBE-IS into ASIS-IS at least for two main reasons: (1) some evolution primitives, used by the evolution, can be irreversible themselves (e.g. the case of an existing integrity rule relaxed by the evolution), and (2) actors, and even a part of the enterprise, can be completely disoriented to go back to TOBE-IS after all the efforts they have done to adapt to ASIS-IS. So, a decision to perform an evolution must be very well prepared to decrease the risks of failure. For this purpose, we explore a generic lifecycle of an evolution, first at atomic primitive level, then at composite primitive level and finally at evolution level.

An atomic primitive can be performed stand-alone in two steps: (1) preparation and (2) execution or abort. They are defined as follows:

- *Preparation*: prepares the *disabling list* of ISIS instances and IS objects to be disabled if success, the *creating list* of ISIS instances and IS objects to be created if success, the list of reports of changes, and the blocking list of ISIS instances;
- *Execution*: sends the reports of changes, blocks the concerned IS parts, disables/creates effectively contents of the disabling/creating lists, then unblocks the blocked IS parts.

The work done at the preparation step serves to decide whether the primitive should be executed or aborted. Finally, the execution of the primitive can succeed or fail. For example, it fails if it cannot block an IS part. As an example let us consider the deletion of a role: its creating list is empty and its disabling list contains all the assignments of operations to this role. Blocking these assignments signifies these operations cannot be executed by means of this role during the deletion of the role. It can fail if an IS actor is working through this role.

In the case of the atomic evolution primitive "Create an instance *Cl* of Class", the preparation step defines:

- how to fill the new class with objects,
- how to position it in the IS schema by linking *Cl* to other IS classes by means of existential dependencies,
- how to alert the managers of Role, Operation and Integrity_Rule about the *Cl* creation.

Besides, it is important to create together with *Cl* its methods and attributes, and even the association classes between *Cl* and other IS classes. For that, we need a more powerful concept, the composite evolution primitive, as presented below.

A composite primitive is composed of other composite or atomic primitives, which builds a hierarchy of primitives. The top composite primitive is at the

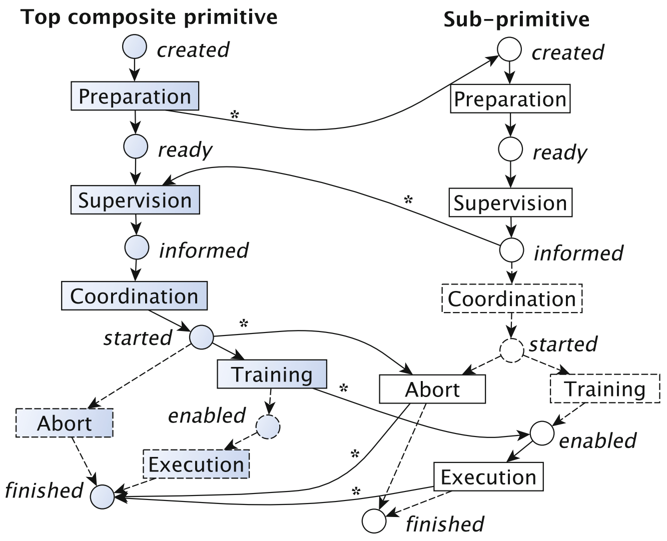

Fig. 5. Coordination of the lifecycle of a composite primitive (left) with the lifecycles of its sub-primitives (right); * indicates multiple transitions, dashed lines indicate that the step is under the responsibility of the lower or upper level model.

root of this hierarchy; the atomic primitives are at its leaves. Every composite primitive has a supervision step, which controls the execution of all its sub-primitives. Only the top composite primitive has in addition a coordination step, which takes the same decision to enable or abort for all its sub primitives in the hierarchy. The main steps of a composite primitive life cycle are:

– *Preparation*: creates all direct sub-primitives of the composite primitive;
– *Supervision*: determines the impacts and the managerial effects from the enabling lists and the creating lists established by the sub-primitives;
– *Coordination*: takes the decision of enabling or aborting primitive processing and transmits it to the sub-primitives;
– *Training*: this is a special step for the top primitive; it concerns training of all actors concerned by the whole evolution. This step is performed thanks to the actors' responsibility spaces.

The top composite primitive is successful if all its sub-primitives are successful; it fails if at least one among its sub-primitives fails. The life cycle of the atomic primitives must be adapted by adding the abort decision and by taking into account that enable/abort decisions are made by a higher level primitive. Figure 5 illustrates the co-ordination between the composite primitive lifecycle and its sub-primitive lifecycle.

Thus, from the atomic evolution primitive “Create Class” we build the composite evolution primitive “C-Create Class” with the following sub-primitives:

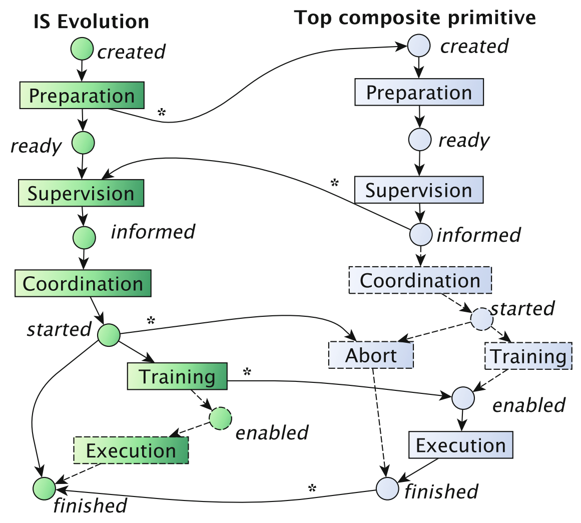

Fig. 6. Coordination of the IS evolution lifecycle with the lifecycles of its top composite primitives

- Create Class, which is used to create the intended class *Cl* and also all the new classes to associate *Cl* with other classes, as mentioned previously,
- Create Class_Concept, Create Class_Attribute,
- if necessary, Create Attribute, and Create Domain,
- C-Create Method with its sub-primitives Create Method and Create Attribute_Method.

Let us now look at the lifecycle of an entire evolution, which is a composition of primitives. During the processing of an evolution, the *preparation* step consists in selecting the list of composite primitives, whose processing will realize this evolution. Then, from the impacts and the managerial effects determined by the supervision steps of these composite primitives, the *supervision* step of the evolution determines a plan for processing these primitives. It decides which primitives can/must be executed in parallel and which in sequence. Next, the *coordination* step launches processing of primitives following the plan. After analyzing their results (success or failure), it decides to launch other primitives and/or to abort some of them. Finally, the evolution is finished and it is time to assess it. Indeed, the evolution processing transforms the enterprise and its ways of working, even if processing of some composite primitives fails. Due to the important complexity, it seems important to place the *training* step at the evolution level and not at the level of composite primitive. Of course, the training step of a composite primitive must be realized before its execution. But, in this way, it is possible to combine training steps of several composite primitives into one, and to obtain a more efficient training in the context of the evolution. Figure 6 shows the coordination between the lifecycles of IS evolution and its top composite primitives.

6 Illustrating Example

To illustrate our approach, we use the example of a hospital. Figure 7 depicts a small part of the kernel of its IS schema. In this example we will consider:

– one organizational unit: the general medicine department,
– two positions: the doctor and the nurse,
– two activities of a doctor: a_1 concerning the care of patients (visit, diagnostic, prescription) and a_2 concerning the management of the nurses working in her team.

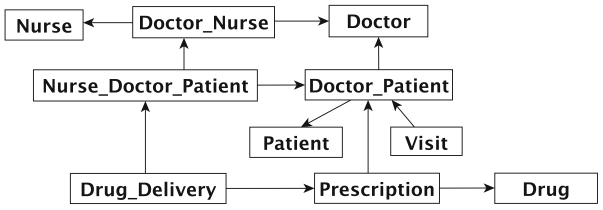

Fig. 7. A small part of the ASIS-IS schema of the hospital

To illustrate an evolution case, let us suppose that now our hospital has to apply new rules for improving patients' safety. To this end, each doctor will be in charge to guarantee that nurses of her team have sufficient competences for administrating the drugs she can prescribe. So, the IS of the hospital must evolve, especially by introducing new classes: Nurse_Drug that associates a nurse with a drug for which she is competent according to her doctor, and Doctor_Drug that associates a doctor with a drug that she can prescribe. The TOBE-IS schema is shown in Fig. 8.

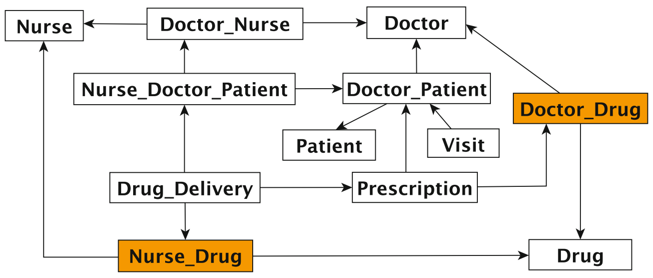

Fig. 8. A part of the TOBE-IS schema of the hospital

The IS evolution is then composed of 2 top composite primitives, one around Doctor_Drug (DD), the other one around Nurse_Drug (ND). The first one is built from the composite primitive C-Create Class to create the instance *DD* of the ISIS element Class. Its preparation step specifies:

- the DD objects will be obtained from the ASIS-IS class Prescription;
- DD will be existentially dependent on the IS classes Doctor and Drug, and Prescription will become existentially dependent on DD and no more directly dependent on Drug;
- the alerts for Role, Activities, Positions, Persons about the changes, especially in the creation an object of Prescription, which in TOBE-IS must be related to a DD object;
- creation of DD objects, creation or modification of roles for reaching them;
- the blocking list for its execution, which includes Doctor, Drug and Prescription.

The second composite primitive is built from the composite evolution primitive C-Create Class to create the instance *ND* of the ISIS element Class. Its preparation step specifies:

- the ND objects will be obtained from the ASIS-IS class Prescription;
- ND will be existentially dependent on the IS classes Nurse and Drug, and Drug_Delivery will become existentially dependent on ND;
- the alerts for Role, Activities, Positions, Persons about the changes, especially in the creation an object of Drug_Delivery, which must be related to a ND object;
- creation of ND objects, creation or modification of roles for reaching them;
- the blocking list for its execution, which includes Nurse, Drug and Drug_Delivery.

In the case of this example, the execution process of the IS evolution after the training of involved actors is simple: to execute the top evolution composite primitives related to Doctor_Drug and then to Nurse_Drug.

7 Conclusion

Handling information systems evolution is a complex task that has to be properly defined, planned and assessed before its actual execution. The result of each IS evolution has impact on the sustainability on organization's ISs and also on the efficiency of the organization's activity. So this task is not only complex but also critical.

In this paper, we continue to present our work on a conceptual framework for IS evolution steering that aims to establish the foundation for the development of an Informational Steering Information System (ISIS). In particular, we dedicate this paper to the engineering aspects of the concept of IS evolution, and present its metamodel, which is one of the components in our framework (Fig. 1).

The role of the IS Evolution Metamodel consists in supporting the operationalization of the IS evolution. Therefore, it includes two views: structural and lifecycle. The structural view allows to progressively decompose a complex IS evolution into a set of atomic primitives going trough several granularity levels of composite primitives. The obtained primitives as robust because they follow generic evolution rules and take into account horizontal and vertical effects on

ISIS and IS. They are also smart because they pay attention to the managerial effects of IS evolution at the human level. The lifecycle view helps to operate IS evolution at its different levels of granularity by providing a set of models and rules for progressing from one step to another.

To complete our framework for IS evolution steering we still need to define the Impact Space component that will provide mechanisms to measure the impact of IS evolution and to take decisions accordingly. With the IS Evolution Metamodel we have prepared the basis for developing the detailed guidance for IS evolution steering, which will complete our work on this conceptual framework.

References

1. Olivé, A.: Conceptual schema-centric development: a grand challenge for information systems research. In: Pastor, O., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 1–15. Springer, Heidelberg (2005). doi:10.1007/11431855_1
2. Opprecht, W., Ralyté, J., Léonard, M.: Towards a framework for enterprise information system evolution steering. In: Frank, U., Loucopoulos, P., Pastor, Ó., Petrounias, I. (eds.) PoEM 2014. LNBIP, vol. 197, pp. 118–132. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45501-2_9
3. Ralyté, J., Opprecht, W., Léonard, M.: Defining the responsibility space for the information systems evolution steering. In: Horkoff, J., Jeusfeld, M.A., Persson, A. (eds.) PoEM 2016. LNBIP, vol. 267, pp. 179–193. Springer, Cham (2016). doi:10.1007/978-3-319-48393-1_13
4. Le Dinh, T.: Towards a new infrastructure supporting interoperability of information systems in development: the information system upon information systems. In: Konstantas, D., Bourrières, J.P., Léonard, M., Boudjlida, N. (eds.) Interoperability of Enterprise Software and Applications, pp. 385–396. Springer, London (2006). doi:10.1007/1-84628-152-0_34
5. IS-SM: IS-SM. http://cui.unige.ch/~ralyte/ISIS/IS-SM.html
6. Pons, C., Kutsche, R.D.: Model evolution and system evolution. In: V Congreso Argentino de Ciencias de la Computacion (CACIC 1999), Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina (1999)
7. Burger, E., Gruschko, B.: A change metamodel for the evolution of MOF-based metamodels. In: Modellierung. LNI, GI, vol. 161, pp. 285–300 (2010)
8. Aboulsamh, M.A., Davies, J.: Towards a model-driven approach to information system evolution. In: Song, W., et al. (eds.) ISD 2009, pp. 269–280. Springer, New York (2009). doi:10.1007/978-1-4419-7355-9_23
9. Kchaou, D., Bouassida, N., Ben-Abdallah, H.: A MOF-based change meta-model. In: The 13th International Arab Conference on Information Technology (ACIT 2012), Zarq, Jordan, pp. 134–141 (2012)
10. Ruiz Carmona, L.M.: TraceME: traceability-based method for conceptual model evolution. Ph.D. thesis. Universitat Politecnica de Valencia (2016)
11. Lehman, M., Fernández-Ramil, J.C.: Software Evolution. Wiley, Hoboken (2006)
12. Lehman, M.M., Ramil, J.F., Kahen, G.: Evolution as a noun and evolution as a verb. In: Workshop on Software and Organisation Co-evolution (SOCE 2000) (2000)
13. Andany, J., Léonard, M., Palisser, C.: Management of schema evolution in databases. In: VLDB 1991, pp. 161–170. Morgan Kaufmann (1991)