Service–Driven Information Systems Evolution: Handling Integrity Constraints Consistency

Nicolas Arni-Bloch, Jolita Ralyté, and Michel Léonard

University of Geneva, CUI, 7 route de Drize, CH-1227 Carouge, Switzerland
{Nicolas.Arni-Bloch,Jolita.Ralyte,Michel.Leonard}@unige.ch

Abstract. Changes and extensions of enterprise information systems (IS) often engender their fragmentation and redundancy. In order to overcome these problems, service-driven IS development is considered as a potential solution to support IS evolution when guaranteeing its integrity and consistency. In this work we consider the integration of new information system services (ISS) into a legacy IS and, in particular, how the consistency of integrity constraints (IC) governing the IS and the new ISS has to be handled in order to guarantee their validation. Five IC handling strategies are proposed in the form of method chunks and the impact of their application is measured with four service evolution indicators.

Keywords: Information system service, service integration, integrity constraint.

1 Introduction

Enterprise Information Systems (IS) are generally built on several applications or components that support business activities. Due to several factors such as business innovation, IT evolution, addition of new applications, these components become heterogeneous and specialized. This situation leads to fragmented IS and therefore to the redundancy between different IS components sharing some common parts. To organize the modularity of the IS and to manage these common parts the notion of information system service is proposed [2, 3].

Current service engineering approaches and architectures like SOA [7, 11, 12] concentrate their study on the rebuilding enterprise IS architecture in terms of autonomous services that can be composed afterwards in different ways. The autonomy principal is reached by reengineering the IS where the modularity of each service is carefully (painfully) defined. In this perspective, services are elaborated from scratch with the objective to avoid any overlap between them. However, in reality the lifecycle of the enterprise IS is a continuous incremental and evolutionary process. It is not possible at each iteration to rethink the entire IS in order to guarantee the autonomy and correctness of the existing and new services. Therefore, we need theories and methods for IS service evolution as proposed in [15] as well as formalisms and indicators to evaluate the impact of the extension of the IS with new services. This impact is, among others, influenced by the granularity and abstraction level of a service, which varies from a simple utility that logs errors to a more abstract complete business process [10, 17].

A. Persson and J. Stirna (Eds.): PoEM 2009, LNBIP 39, pp. 191–206, 2009.

Depending on service granularity and abstraction level, its integration into an existing IS is more or less complex. In this work we focus our attention on services with a rather high level of modularity each of them representing a work unit with a precise semantic; we call them information services or Information System Services (ISS). An ISS provides the information space and capabilities to the actors that have the responsibility to use them in order to perform their daily activities restrained by the regulation polices. When the granularity of an ISS is high, the overlap with other IS services becomes hard to avoid. Some service normalization techniques [7] exist to limit the functional redundancy. However, in order to guarantee IS data and process quality it is important to handle the overlap of its services by consolidating their data, capabilities, rules and responsibilities and specifying their cooperation strategy.

While integration of IS data schemas and models was largely discussed in the literature [4, 16, 19], other IS spaces (dynamic, rules and responsibilities) had less attention. In this work, we discuss the consolidation of the IS rule space and handling integrity constraints (IC) consistency when integrating new ISS into a legacy IS. The topic of IC validation has been studied for many years [9, 8] and is closely related to the transaction theories [6, 14]. Our aim here is not to propose a yet another way to ensure the integrity of data or transactional properties but rather to take the orientation of service change management and to study how handling the IC validation can affect the modularity and autonomy of the ISS. ICs are the basis for guaranteeing the information quality and the business rules preservation. They are also an important cause of dependency between different ISS and thus are in tension with the loose coupling and autonomy principles of the conventional SOA approach [7, 11].

In the following we define and illustrate the notion of ISS (section 2) and formalize the overlap which can appear when integrating new ISS into a legacy IS (section 3). Section 4 discusses the generic overlap situations and proposes five method chunks for handling IC consistency. Evaluation of the impact when applying these method chunks is discussed in section 5 and section 6 concludes our paper.

2 Information System Service

An *ISS* is a component of an information system representing a well defined business unit that offers capabilities to realise business activities and owns resources (data, rules, roles) to realize these capabilities. Formally, an ISS is defined as follows:

Definition 1 (Information System Service). An Information System Service is an autonomous coherent and interoperable component of an information system composed of four spaces: static, dynamic, rule and role: ς=<*sSs, sDs, sRs, sOs*> where*:*

- *sSs(ς): {Class}*, represents the set of classes of the service ς,
- *sDs(ς): {Action}*, represents the set of actions defined by the ser*vice* ς,
- *sRs(ς): {Rule}*, represents the set of rules that govern the service ς,
- *sOs(ς): {OrganizationalRole}*, represents the set of organizational roles that have rights and responsibilities on the service ς.

Fig. 1 illustrates an ISS, named *DiplomaManagementService (DMService)*, providing diploma management capabilities for a University. In fact, this service is extracted

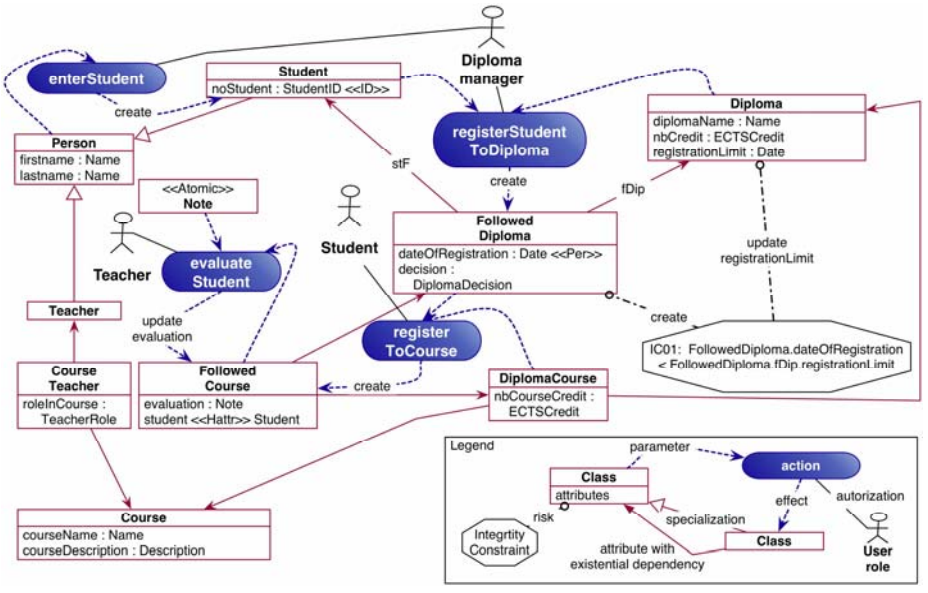

Fig. 1. Partial specification of the DiplomaManagementService (DMService)

from the effective IS of the University of Geneva. For the reason of readability the models used in this paper are simplified. As shown Fig. 1, the four ISS spaces are represented in the same model.

The static space of an ISS embodies its data structure and is represented by a class diagram. Naturally, the main concepts defined in this space are: *class* (e.g. *Student*, *Diploma*), *attribute* (e.g. *diplomaName*), *key* (e.g. *noStudent*) and *method* (not shown in Fig. 1). In order to guarantee the consistency of the data model we limit our model to only two types of relationship, *existential dependency* (e.g. the existence of an object of the class *Followed Diploma* depends on the existence of one object of the class *Student* and one object of the class *Diploma*) and *specialization* (e.g. the class *Student* is a specialisation of the class *Person*).

The dynamic space of an ISS represents the behaviour of service capabilities. For example, the *DMService* (Fig. 1) offers several capabilities: to create the curriculum of each diploma by defining courses and linking them to their teachers, to manage students' registration to different diplomas and to the corresponding courses as well as their examination. The main concepts of this space are *action* and *effect*. An action (e.g. *registerStudentToDiploma*) provokes an effect (e.g. *create* an object of the class *Followed Diploma*) during the execution of its process. The notion of effect is used to characterize the result of the action and allows to evaluate the impact of the action on the rule space.

Definition 2 (Effect). An effect is a tuple < *pr, target* > where *pr* defines the kind of effect from the set of primitives *pr* ∈ *{create, enter, exit, delete, update, list, read, return, call}* and *target* is either a class or a class and a set of attributes or an action.

$sDs_{effects}(\varsigma)$: *{effect}* represents the set of effects that the service ς can generate when executing its actions.

The objective of the rule space is to preserve the correctness and consistency of the ISS during its exploitation. Two types of rules, *conditions* and *integrity constraints*, have to be considered. While conditions regulate the execution of service actions, the role of integrity constraints is to ensure the integrity of service data. In this paper we focus our attention on the integrity constraints. Fig. 1 illustrates one of the *DMService's* integrity constraints named *IC01* which restricts the period of students' registration to a diploma – the date of effective registration has to be inferior to the predefined registration limit date.

Definition 3 (Integrity constraint). An integrity constraint (IC) is a rule that has to be verified in each state of the service or at each modification of it. Given a service ς, and an IC $ic \in sRs(\varsigma)$, the classes and attributes that participate in the validation of the *ic* define its validation context *(context(ic): {class})*. The *ic* also has a *scope(ic): {effect}* which includes all the effects that could transgress this rule. Each effect is called a *risk* of the IC.

In order to be able to validate the IC defined in its rule space *(sRs)*, the ISS has to know all the classes defined in each rule's context. For this purpose we define the *rule completeness* that aims to ensure that the ISS is defined on the static space *(sSs)* that offers all the information needed by its rules.

Rule completeness. Given a service ς, $\forall rule \in sRs(\varsigma) : context(rule) \subset sSs(\varsigma)$ where *context(rule)* is a set of classes needed for the evaluation of the rule, i.e. the context of the rule.

Finally, the role space of an ISS defines the organizational roles and their rights and responsibilities on the service. A role is an element of an organization that has responsibilities in achieving activities to reach a common objective of the organization. In our example (Fig. 1), the main roles using the *DMService* capabilities are *Student*, *Teacher* and *Diploma manager*. More details about the notion of ISS and its metamodel MISS can be found in [3][1].

3 Defining ISS Overlap and Inconsistency

IS evolution in service-driven perspective means that new ISS are integrated into the legacy IS. Integrating two ISS will certainly create overlap situations in different spaces (shared data, duplicated actions, conflicting rules and responsibilities) that have to be handled in order to ensure each service consistency and also to preserve their modularity and autonomy. Let us consider the *DMService,* discussed in the previous section, as a legacy one which has to be extended with a new service supporting on-line registration of students to the University named *UniversityRegistrationService (URService)*.

[1] The role space is added to the original ISS metamodel MISS presented in [3].

Fig. 2 illustrates the new *URService* (simplified for the reason of readability) to be added to the Unversity IS. This service publishes two actions as public methods: *OnlineRegistration* and *RegisterTo-University*. The first action allows to create a *UniversityRegistrationRequest* on the web. It is a complex process (not detailed here) that builds the registration including different required documents according to the integrity constraints defined on the *UniversityRegistrationRequest*. The second action is dedicated to validate the on-line created registration request by the administration, to record the corresponding person as a student and to register him/her to a diploma.

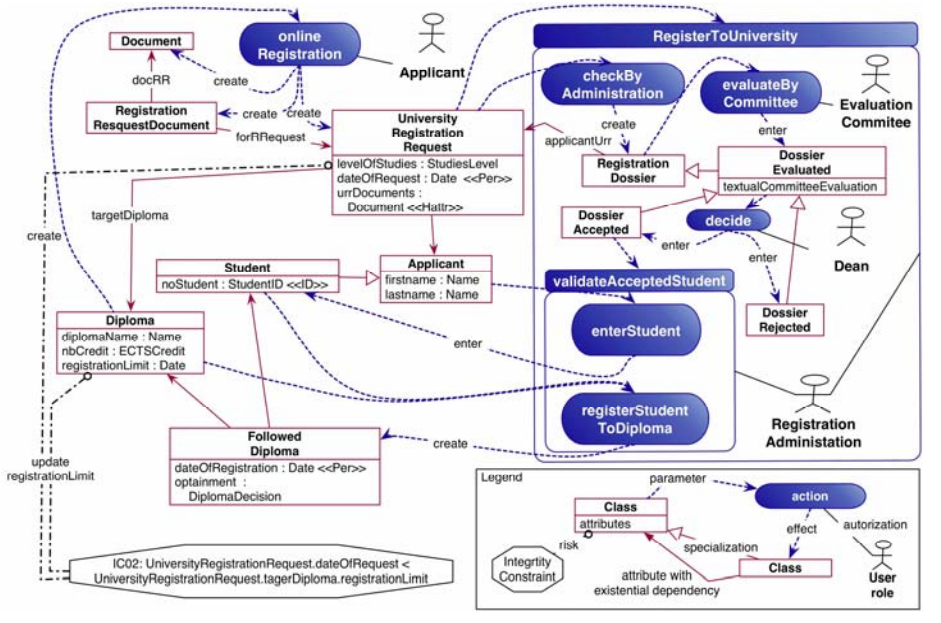

Fig. 2. Partial specification of the new UniversityRegistrationService (URService)

This case illustrates several overlap situations between the two services, the legacy and the new one. In fact, we can see that there is a static overlap between classes *Person*, *Student*, *Diploma* and *FollowedDiploma* as well as a dynamic overlap between actions *registerStudentToDiploma*. If an object, class or action belongs to several services of the same IS, some redundancy of information and inconsistency of rules and responsibilities can be expected and, therefore, has to be managed. In this section we define the notion of overlap which appears in different ISS spaces when integrating an ISS into an IS. Because of the lack of space we limit the number of definitions to those necessary to handle rule space inconsistencies.

We say that a class is in overlap if it is used in the definition of the static space of more than one service. It means that the definition of the given information is used by several services to offer their capabilities.

Definition 4 (Class overlap). Given a class *cl*, we say *cl is in overlap if* $\exists \varsigma, \varsigma' \; \varsigma \neq \varsigma' : cl \in sSs(\varsigma) \wedge cl \in sSs(\varsigma')$ where $sSs(\varsigma)$ represents the static space of the service ς, i.e. its set of classes.

The role that a class plays in a service is specified by the effects that this service can cause on the class. For example, some services can only access the objects of the class in overlap but cannot change their state; other services can only create new objects of the class but cannot modify or delete them. In order to refine the notion of overlap, we define the *effect overlap*. An effect is in overlap if it can be generated by more then one service. It means that several services share the same responsibility on the information represented by the class. For example, the effect <*create, an object of the class FollowedDiploma*> can be generated by both services (Fig. 1 and Fig. 2) and therefore it is in overlap.

Definition 5 (Effect overlap). Given an effect *ef*, we say *ef is in overlap if* $\exists \varsigma, \varsigma' \; \varsigma \neq \varsigma' : ef \in sDs_{effects}(\varsigma) \wedge ef \in sDs_{effects}(\varsigma')$. $sDs_{effects}(\varsigma)$ represents the set of effects that the service ς can generate.

A rule is in overlap if some classes of its context are in overlap. It means that some part of the validation context of the rule is shared by several services. In the case of IC, it is essential to consider the effects in overlap that could violate this IC. For that, we define the *IC risk overlap*. For example, the context of the *IC02* (Fig. 2) defined in the rule space of the *URService* contains the class *Diploma* which is also belongs to the static space of the service *DMService*. Therefore, this rule is in overlap.

Definition 6 (Rule overlap). Given a rule *r*, we say *r is in overlap if* $\exists cl: cl \in context(r) \wedge cl$ *is in overlap.*

Definition 7 (Integrity constraint risk overlap). Given integrity constraint *ic*, we say *ic is in risk overlap if* $\exists ef: ef \in scope(ic) \wedge ef$ *is in overlap.*

An IC in overlap can be either a part of each service for which it is in overlap, or belong to only some of the services and be missing in the others. In the first case, all services guarantee the handling of the IC and the consistency of the IS regulatory policy is assured. In the second case, as the IC is not known by some services and the IS regulatory policy can be transgressed. We should therefore harmonize this policy by identifying the inconsistencies in the IC overlap.

Definition 8 (Integrity constraint overlap inconsistency). An integrity constraint *ic* has an overlap inconsistency with a service ς *if* $\varsigma \in Risk\text{-}OSS(ic) \wedge ic \notin sRs(\varsigma)$ where the *Risk-OSS(ic)* (named risk overlap service set) is the subset of IS services that include at least one effect from the scope of the *ic: Risk-OSS(ic)* $= \{\varsigma : \exists ef, ef \in scope(ic) \wedge ef \in sDs_{effects}(\varsigma) \wedge$ *ef is in overlap}.*

An IC overlap inconsistency appears when the IC is in risk overlap with a service but is not defined in its rule space. From the legacy IS point of view, this inconsistency leads to a service that is not governed by the concerned IC. The service can therefore transgress it without knowing about its existence, the IS rules policies can be compromised and the IS consistency is not guaranteed. From the new ISS point of view,

the process of the service can be stopped by an external IC unknown to the service. In this case, the service does not have the necessary information to handle this violation. The actors using the service are unable to perform the work because the service does not provide the required knowledge to resolve the violation of the IC. Several strategies can be defined to resolve this kind of inconsistency. We discuss them in the next section.

4 Handling Integrity Constraints Consistency

Integration of a new ISS into a legacy IS mainly consists in identifying and characterising the overlap in the four information spaces (static, dynamic, rules and roles), handling it (modifying, merging, adding or removing elements in the four spaces) and consolidating the integrated specifications [2][2]. In this paper we focus our attention on the rule space overlap handling and propose five IC consistency handling strategies formalised in the form of method chunks [13]. Selection of the appropriate method chunk depends on the overlap situation and the impact that the application of the chunk results on the concerned service.

4.1 Integrity Constraints Overlap Situations

Because of the static space overlap occurring when integrating a new ISS into a legacy IS some inconsistencies in IC overlap can appear. We identify two generic integrity constraints overlap situations – *total* and *partial* – that generate corresponding inconsistencies and have to be resolved during the ISS integration.

Total IC overlap (see Fig. 3a) appears when the context of an IC (the collection of classes required for this rule validation) defined in one service totally belongs to the static space of the other service. If the IC is in overlap inconsistency with the service (i.e. the IC is not defined in the rule space of the service) we say that this IC is in *total overlap inconsistency* with the service. In this situation, the service owns the required information to handle the IC (i.e. to validate and deal with the potential violation) and the inconsistency can be resolved by simply adding the IC to the service rule space; no evolution of the static space is required.

Definition 9 (Integrity constraint total overlap inconsistency). An integrity constraint *ic* has a total overlap inconsistency with a service ς if *ic is in overlap inconsistency with the service* $\varsigma \wedge context(ic) \subseteq sSs(\varsigma)$.

Partial IC overlap (see Fig. 3b) appears when the context of an IC defined in one service partially belongs to the static space of the other service (not all the classes of the IC context exist in the static space of the service). We say that this IC is in *partial overlap inconsistency* with the service. In this situation, the service does not own enough information to handle the IC and it is not possible to simply add this IC to the service rule space; some evolution of the static space of the service has to be done, or the IC has to be modified.

[2] In the paper [2] we have proposed a process model for IS integration into a legacy IS.

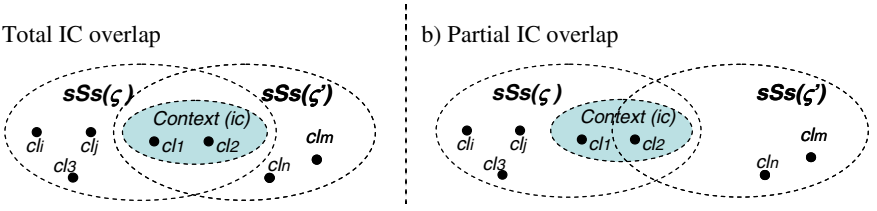

Fig. 3. Generic IC overlap situations

Definition 10 (Integrity constraint partial overlap inconsistency). An integrity constraint *ic* has a partial overlap inconsistency with a service ς *if ic is in overlap inconsistency with the service* $\varsigma \wedge context(ic) \cap sSs(\varsigma)$.

In our case, the *IC01* and *IC02* illustrated in Fig. 1 and Fig. 2 respectively are examples of integrity constraints in overlap inconsistency. The *IC01* defined in the *DMService* is in total overlap inconsistency with the *URService* because its context *{Diploma, FollowedDiploma}* is included in the static space of the *URService*. The *IC02* defined in the *URService* is in partial overlap inconsistency with *DMService* because its context *{Diploma, UniversityRegistrationRequest}* partially belongs to the static space of *DMService*. In both situations, the IC overlap has to be resolved – the missing rule has to be added into the corresponding service rule space and/or different elements of the service have to be modified. In order to validate an IC, the service containing it needs to have access to all the classes of the IC context. It means that the introduction of a new IC can require some evolution of the concerned service static space. In the contrary, if the decision is do not include the IC into the rule space of the service the overlap has to be settled by other means, for example by reducing service responsibility or by transforming the IC into a simple condition. For each situation we define several strategies to handle IC inconsistencies that we present in the form of method chunks[3].

4.2 Method Chunks for Integrity Constraints Consistency Handling

The collection of method chunks presented below is representative but not exhaustive; we aim our approach to be progressive and easily extensible with new method chunks and therefore to follow the situation-driven way of thinking.

Simple IC Addition Strategy. To deal with the total IC overlap inconsistency we propose to add the missing IC into the rule space of the corresponding service. It is clear that the rule completeness has to be validated during such an operation. We call this strategy *simple IC addition strategy* and the method chunk supporting it is illustrated in Table 1. Because all the classes of the IC context are already included in the service static space, the service has enough information to handle this IC. The service has to be updated to take into account this new rule and the actors using this

[3] In [13] a method chunk is defined as an autonomous and coherent part of a method supporting the realisation of some specific IS development activity.

service have to be informed about the new rule but the addition of the IC does not require any evolution of the static space of the service. An example of this situation is the *IC01* (see Fig. 1) that is in total overlap inconsistency with the *URService*. This service has actions (e.g. *validateAcceptedStudent*) that cause effects which are risks for the *IC01*. To resolve this inconsistency the *IC01* can be added to the *URService* without any modification of the static space of this service.

Table 1. Method chunk for IC consistency handling following simple IC addition strategy

Chunk name	Simple IC addition
Situation	< Integrity Constraint *ic*, Service ς >
Preconditions	c1) $context(ic) \subset sSs(\varsigma)$ c2) $ic \notin sRs(\varsigma)$
Intention	Settle the integrity constrain *ic* overlap with the service ς following simple IC addition strategy
Postconditions	c1) $ic \in sRs(\varsigma)$
Guideline	1) Add the integrity constraint *ic* to the service ς. 2) Verify that all the actions of ς that have effects included in the scope of *ic*, validate the constraint.

Service Static Space Extension Strategy. In the case of partial IC overlap inconsistency, it is not sufficient to add the IC to the service rule space – the missing classes from the IC context have to be added into the service static space in order to respect the completeness of the rule. It means that the static space of the service is extended with new classes. We call this strategy *static space extension* and the method chunk supporting it is illustrated in Table 2. For example, the *IC02* (Fig. 2) is in partial overlap inconsistency with the *DMService*. The *DMService* does not have enough information to handle this rule ($context(IC01) \not\subset sSs(DMService)$) but can violate it. The *UniversityRegistrationRequest* class can be added to its static space in order to resolve the inconsistency.

Table 2. Method chunk for IC consistency handling by extending the static space of the service

Chunk name	Service static space extension
Situation	< Integrity Constraint *ic*, Service ς>
Preconditions	c1) $context(ic) \not\subset sSs(\varsigma)$ c2) $ic \notin sRs(\varsigma)$
Intention	Settle the integrity constrain *ic* overlap with the service ς by extending the static space *sSs* of the ς with classes from the *ic* context
Postconditions	c1) $ic \in sRs(\varsigma)$ c2) $context(ic) \subset eSs(\varsigma)$
Guideline	For each classe $cl \in context(ic)$: if $cl \notin sSs(\varsigma)$: add *cl* to the static space of the ISS (add to sSs(ς)) add ic to the service by applying the method chunk "Simple rule addition".

Service Responsibility Reduction Strategy. Another way to deal with the IC overlap (total or partial) is by reducing the responsibility of the service. In the contrary to the previous cases, the IC is not added into the rule space of the service but the responsibility

of the service is modified in order to handle the IC overlap inconsistency. To reach this objective, it is necessary to remove from the service all the actions that have effects which could violate the IC. By doing that, the responsibility of the service is decreased as well its capability. We name this strategy *responsibility reduction* and the method chunk is presented in Table 3. In our example, applying this method chunk on the *IC02* and the *DMService* would require to remove from the *DMService* the responsibility of updating the objects of the class *Diploma*, more exactly to update its attribute *registrationLimit*.

Table 3. Method chunk for IC consistency handling by reducing service responsibility

Chunk name	Service responsibility reduction
Situation	< Integrity Constraint *ic*, Service ς >
Preconditions	c1) $context(ic) \cap sSs(\varsigma) \neq \varnothing$ c2) $ic \notin sRs(\varsigma)$
Intention	Settle the integrity constrain *ic* overlap with the service ς by reducing the responsibility of the ς
Postconditions	c1) $scope(ic) \not\subset sDs_{effect}$
Guideline	For each action $act \in sDs(\varsigma)$: if $\exists ef_i \in effects\ (act), ef_i \in scope\ (ic)$: remove *act* from $sDs(\varsigma)$.

Control Extension Strategy. This method chunk (see Table 4) aims to guarantee the non-violation of an IC in partial or total overlap without adding this IC to the service and by preserving the responsibility of the service. This objective is reached by the introduction of an overlap protocol defined with new IC having a more restrained scope. This allows to maintain the IC without increasing the information coupling and without decreasing the responsibility of the service. Meanwhile, the service will be more constrained by the new rules. The strategy is named *service control extension.* In the case of the *IC02*, we can add a state to the *Diploma* class, as for example *DiplomaInElaboration* and *DiplomaInProduction*. The *DMService* can modify the attribute *registrationLimit* while a diploma is in the *elaboration state*, but cannot modify it when a diploma is in the *production state*. In the *URService* side, the registration is allowed only to diplomas that are in production and therefore the *IC02* cannot be violated by the *DMService*.

Table 4. Method chunk for IC consistency handling with service control extension strategy

Chunk name	Service Control Extension
Situation	< Integrity Constraint *ic*, Service ς >
Preconditions	c1) $context(ic) \cap sSs(\varsigma) \neq \varnothing$ c2) $ic \notin sRs(\varsigma$
Intention	Settle the integrity constrain *ic* overlap with the service ς by adding a control element to the service ς
Postconditions	c1) *ic* cannot be violated by any action of ς
Guideline	Add new rules and eventually new elements (attribute, classes) that guarantee that *ic* cannot be violated by any action of theς. Eventually modify the *ic*.

IC Transformation into a Condition Strategy. This method chunk (see Table 5) transforms an IC into a condition and offers therefore the possibility to choose the actions that have to handle this condition. By transforming the IC into a condition, the validation of the rule is not anymore mandatory and the service with which it was in overlap does not need to validate the rule. In the case of the *IC02*, transforming this constraint into a condition will allow to validate *IC02* only by the actions of the *URService*. Registration to diploma through the *URService* will follow the *IC02* condition while the *DMService* will not. This strategy decreases the global level of consistency of the IS by transforming the IC into a simple condition.

Table 5. Method chunk for IC consistency handling by IC transformation strategy

Chunk name	IC transformation into a condition
Situation	< Integrity Constraint ic, Service ς >
Preconditions	c1) $context(ic) \cap sSs(\varsigma) \neq \varnothing$ c2) $ic \notin sRs(\varsigma)$
Intention	Settle the integrity constrain *ic* overlap with the service ς by transforming the *ic* into a condition
Postconditions	c1) *ic* is a condition
Guideline	Transform *ic* into a condition.

5 Evaluating Integrity Constraints Consistency Handling Impact

The method chunks presented in the previous section demonstrate that the impact of the service integrity constraints consistency handling depends on the selected strategy and can be more or less cumbersome to deal with. We propose here a set of indicators in order to support the evaluation of this impact and the selection of the most appropriate method chunk in each situation.

5.1 Service Evolution Indicators

For this study we select four properties of a service, named *needs* ($\mathcal{N}$), *capabilities* ($\mathcal{CAP}$), *overlap* ($\mathcal{O}$) and *consistency* ($\mathcal{CO}$), and we aim to assess the impact of each IC consistency handling strategy on these properties (i.e. indicate how they evolve).

The evaluation of the evolution of service *needs* consists in detecting if a given method chunk increases or reduces the information necessary for the execution of service capabilities. The evaluation of service *capabilities* property means to identify if a method chunk will cause a loose or an increase of the capabilities offered by the service. Similarly, we evaluate if it will increase or decrease the static space *overlap* of a given service. Finally, the *consistency* property allows the evaluation of the rule system of the service – to measure if the applied strategy will relax or harden the integrity constraints, i.e. more integrity constraints will be defined or some existing integrity constraints will not be validated anymore by the service.

In order to evaluate the impact of the method chunks on the discussed properties we define property evolution indicators. This impact is deterministic when it can be established before the enactment of the method chunk, otherwise it is unknown (?).

Definition 11 (Indicator $\mathcal{I}^{\theta}$). Given a service ς, an indicator $\mathcal{I}^{\theta}$ measures the evolution of its property $\theta \in \{\mathcal{N}, \mathcal{CAP}, \mathcal{O}, \mathcal{CO}\}$ between service variants v_i^{ς} and v_t^{ς}. This evolution can be *an increase (↗), a decrease (↘)* or *stay constant (→).*

The evaluation of the evolution of service needs after application of a method chunk is based on the number of classes in its static space (sSs), which can stay constant or have more or less classes.

Definition 12 (Needs indicator $\mathcal{I}^{\mathcal{N}}$). Given a service ς and its variants v_i^{ς} and v_t^{ς}, we measure the impact on service needs with $\Delta sSs = |sSs(v_t^{\varsigma})| - |sSs(v_i^{\varsigma})|$. The semantic of the needs indicator $\mathcal{I}^{\mathcal{N}}(v_i^{\varsigma}, v_t^{\varsigma})$ is defined as: $\mathcal{I}^{\mathcal{N}} = \rightarrow \Leftrightarrow \Delta sSs = 0$; $\mathcal{I}^{\mathcal{N}} = \nearrow \Leftrightarrow \Delta sSs > 0$; $\mathcal{I}^{\mathcal{N}} = \searrow \Leftrightarrow \Delta sSs < 0$.

For example, the addition of the *IC02* into the rule space of the *DMService* following the *Service static space extension strategy* will result an extension of the static space of the *DMService* with the class *UniversityRegistrationRequest.* It means that more information will be required to realise the capabilities of this service: the *DMService* will have to access the *UniversityRegistrationRequest* class in order to validate the *IC02* (to ensure rule completeness). Moreover, it is not only a question of rule validation, but also a question of information space modification for the actors using this service. After the addition of this new IC, if an actor will need to modify the attribute *Diploma.registrationLimit* he/she will have to know the *IC02* as well as the new class *UniversityRegistrationRequest.*

The evaluation of service capabilities evolution is based on the actions that it can execute. With the static space constant, the decrease of $\mathcal{I}^{\mathcal{CAP}}$ means that the service offers fewer capabilities with the same amount of information.

Definition 13 (Capabilities indicator $\mathcal{I}^{\mathcal{CAP}}$). Given a service ς and its variants v_i^{ς} and v_t^{ς}, we measure the impact on service capabilities with $\Delta sDs = |sDs(v_t^{\varsigma})| - |sDs(v_i^{\varsigma})|$. The semantic of the capabilities indicator $\mathcal{I}^{\mathcal{CAP}}(v_i^{\varsigma}, v_t^{\varsigma})$ is defined as: $\mathcal{I}^{\mathcal{CAP}} = \rightarrow \Leftrightarrow \Delta sDs = 0$; $\mathcal{I}^{\mathcal{CAP}} = \nearrow \Leftrightarrow \Delta sDs > 0$; $\mathcal{I}^{\mathcal{CAP}} = \searrow \Leftrightarrow \Delta sDs < 0$.

If the capabilities indicator decreases the unity of work represented by the service loses some responsibility on the information space. Such a change can result in the incapacity of realising some work with this service, or the work will be harder to do. For example, another possibility for handling the *IC02* overlap with *DMService* is by applying the *Service responsibility reduction* method chunk. In this case, we remove from the *DMService* the possibility to update the registration limit of a diploma (effect <*update, Diploma.registrationLimit*>) and therefore all the actions that can produce this effect. By doing this, we reduce the responsibility space of this service and the actors managing the diplomas will not be able to update this attribute anymore.

The evaluation of the service overlap property is based on the number of classes in the static space that are shared with others services. With a constant number of classes, the increase of service overlap means that some parts of the service that were only used by the service are now shared with other services. In fact, as the overlap is

between at least two services, the shared static space will evolve in more then one service.

Definition 14 (Overlap indicator $\mathcal{I}^{\mathcal{O}}$). Let's $sSs_{shared}(\varsigma)$ represents the subset of the classes of $sSs(\varsigma)$ that are in overlap with other services. $sSs_{shared}(\varsigma)=\{cl \in sSs(\varsigma)$: *cl is in class overlap}* (cf. definition 4). Given a service ς and its variants v_i^{ς} and v_t^{ς}, we measure the impact on service overlap with $\Delta sSs_{shared}= /sSs_{shared}(v_t^{\varsigma})/ - /sSs_{shared}(v_i^{\varsigma})/$. The semantic of the overlap indicator $\mathcal{I}^{\mathcal{O}}(v_i^{\varsigma}, v_t^{\varsigma})$ is defined as $\mathcal{I}^{\mathcal{O}}=\rightarrow \Leftrightarrow \Delta sSs_{shared} = 0$; $\mathcal{I}^{\mathcal{O}}=\nearrow \Leftrightarrow \Delta sSs_{shared} > 0$; $\mathcal{I}^{\mathcal{O}}=\searrow \Leftrightarrow \Delta sSs_{shared} < 0$.

For example, adding the *IC02* to the *DMService* by applying the *service static space extension strategy* implies to add the *UniversityRegistrationRequest* class (from the *URService*) to the *DMService*. The consequences for the *URService* and the *DMService* will be an increase of their overlap.

Finally, the last indicator evaluates the preservation of service consistency. It measures if the service will be more or less constrained after applying a method chunk. Due to the difficulty to reason with integrity constraints in its full generality [18], the evaluation of the level of consistency cannot be reduced to the calculation of the number of service rules.

Definition 15 (Consistency indicator $\mathcal{I}^{\mathcal{CO}}$). Given a service ς and its variants v_i^{ς} and v_t^{ς}, the consistency indicator $\mathcal{I}^{\mathcal{CO}}(v_i^{\varsigma}, v_t^{\varsigma})$ decreases ($\mathcal{I}^{\mathcal{CO}}=\searrow$) when some IC are removed or relaxed from the $sRs(\varsigma)$, it increases ($\mathcal{I}^{\mathcal{CO}}=\nearrow$) when IC are added or hardened, and finally it is stable ($\mathcal{I}^{\mathcal{CO}}=\rightarrow$) when IC stay unchanged.

For example, the *IC transformation into a condition* method chunk transforms the *IC02* into a condition in order to validate it only in the *URService*. This transformation can be seen as an elimination of the IC, or as a relaxing of the IC into a condition. Anyway, the service rule space will be less constrained and therefore the consistency of *URService* will decrease.

5.2 Applying the Indicators

Table 6 overviews the impact of the application of the five IC consistency handling method chunks measured with the four service evolution indicators.

Table 6. Overview of the IC consistency handling impact

Method chunk \ Indicator	Needs $\mathcal{I}^{\mathcal{N}}$	Capabilities $\mathcal{I}^{\mathcal{CAP}}$	Overlap $\mathcal{I}^{\mathcal{O}}$	Consistency $\mathcal{I}^{\mathcal{CO}}$
Simple IC addition	→	→	→	↗
Static space extension	↗	→	↗	↗
Responsibility reduction	→ or ↘	↘	→	→
Control extension	→ or ↗[4]	→	→	↗
IC transformation	→	→	→	↘

[4] The static modularity can increase if the overlap protocol requires new classes.

We can see in this table that the method chunks supporting *simple IC addition* and *IC transformation* strategy have impact only on service consistency; however, while the first increases it the last decreases which can be rather negative especially when the service is a legacy one. Service capabilities evolve only when the *responsibility reduction* method chunk is applied. Reducing service capabilities of a legacy service can be problematic because it concerns an already established work unit. The overlap increases only because of the *static space extension* which means that the autonomy of the involved services is reduced. The application of the *static space extension* method chunk also increases service needs. Adding new classes into the static space of a legacy service is not always appropriate. For example, addition of the *IC02* to the *DMService* following this strategy requires to add the *UniversityRegistrationRequest* class to the *DMService* which is a legacy service. Adding this class (which manages university registration requests) to the service (which manages diplomas) does not make sense from the semantic and responsibility perspective of this service. Finally, if the IC is in overlap with several services, *control extension* strategy can be quite hard to apply as all the involved services will have to implement the overlap protocol.

To summarize, we claim that the objective of these indicators is not to state that one strategy is better than another but rather to indicate the type of impact that should be expected and to help decide which type of impact could be accepted and managed. In general, the impact on a legacy service is always more problematic than the impact on a new one because some conformance invariants, as defined in [1], can forbid some evolution and cause modifications of existing data and processes.

6 Conclusion

Handling information service integration in order to extend an existing IS can be quite cumbersome especially when dealing with service rule space overlap. However, this task is necessary in order to ensure IS policies and integrity on the one hand and to preserve each service modularity and consistency on the other hand. Based on our previous work where we define the notion of ISS [3] and propose a process model for ISS integration into a legacy IS [2], we focus here our attention on the consistency handling of the ISS and IS rule spaces. For this purpose, we define the notion of service rule space overlap and identify generic integrity constraints overlap situations and inconsistencies to be resolved.

The inconsistency of the integrity constraints, which are the most important rules to handle, can be considered in several ways each of them having a different impact on service properties. In this work we propose five IC consistency handling strategies captured in the form of method chunks and we illustrate their application. In order to support the selection of the most appropriate method chunk in a given situation, we analyze the impact that each of them has on four service properties: the *needs* for executing service capabilities, the *capabilities* provided by the service, the static space *overlap* with other services and the *consistency* of the service. As a consequence, we define four service evolution indicators, one for each property. Each indicator shows how the corresponding property can evolve when applying the method chunk and therefore helps to evaluate the impact of its application. It is clear that important changes have to be avoided on the legacy services.

Currently, we focus our effort on identifying and evaluating other method chunks having less impact on service properties, investigating the potential of the service evolution indictors and developing a tool supporting our approach.

The formalization presented in this paper is based on the notion of information service as defined in MISS [3]. However, the proposal can be generalized to other services metamodels such as: WSDL [20], WSPER [5] or Service specification reference model [1]. These metamodels already define the notions of action (named operation or method) and information model (object-oriented or some kind of type system). To apply our approach on these metamodels, we need to extend them with the notion of rule and to annotate the operation or method with the notion of effect. With this pivot concept we are able to identify the actions that can invalidate the IC. We are therefore looking for the possibility to extract service specification knowledge from existing enterprise service registry and to enhance it with the rule knowledge.

References

1. Andrikopoulos, V., Benbernou, S., Papazoglou, M.P.: Managing the evolution of service specifications. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 359–374. Springer, Heidelberg (2008)
2. Arni-Bloch, N., Ralyté, J.: Service-Oriented Information Systems Engineering: A Situation-Driven Approach for Service Integration. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 140–143. Springer, Heidelberg (2008)
3. Arni-Bloch, N., Ralyté, J.: MISS: A Metamodel of Information System Service. In: Proc. of the 17th Int. Conf. on Information System Development (ISD 2008), Paphos, Cyprus (2008)
4. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM Comput. Surv. 18(4), 323–364 (1986)
5. Dubray, J.-J.: Wsper an abstract soa framework. Technical report (2007), `http://www.wsper.org`
6. Elmagardmid, A.K. (ed.): Database Transaction Models for Advanced Applications. Morgan Kaufmann Publishers Inc., San Francisco (1990)
7. Erl, T.: SOA Principles of Service Design. Prentice Hall PTR, Englewood Cliffs (2007)
8. Fahrner, C., Marx, T., Philippi, S.: Dice: Declarative integrity constraint embedding into the object database standard odmg-93. Data Knowl. Eng. 23(2), 119–145 (1997)
9. Grefen, P.W.P.J., Apers, P.M.G.: Integrity control in relational database systems - an overview. Data Knowl. Eng. 10, 187–223 (1993)
10. Haesen, R., Snoeck, M., Lemahieu, W., Poelmans, S.: On the definition of service granularity and its architectural impact. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 375–389. Springer, Heidelberg (2008)
11. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR, Englewood Cliffs (2004)
12. MacKenzie, M., et al.: Reference model for service oriented architecture 1.0. Technical report, Oasis (2006)
13. Mirbel, I., Ralyté, J.: Situational Method Engineering: Combining Assembly-Based and Roadmap-Driven Approaches. Requirements Engineering 11(1), 58–78 (2006)
14. Papazoglou, M.P.: Web services and business transactions. World Wide Web 6(1), 49–91 (2003)

15. Papazoglou, M.P.: The challenges of service evolution. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 1–15. Springer, Heidelberg (2008)
16. Park, J., Ram, S.: Information systems interoperability: What lies beneath? ACM Trans. of Information Systems 22(4), 595–632 (2004)
17. Quartel, D.A.C., et al.: Cosmo: A conceptual framework for service modelling and refinement. Information Systems Frontiers 9(2-3), 225–244 (2007)
18. Queralt, A., Teniente, E.: Decidable reasoning in UML schemas with constraints. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 281–295. Springer, Heidelberg (2008)
19. Quix, C., Kensche, D., Li, X.: Generic schema merging. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 127–141. Springer, Heidelberg (2007)
20. W3C. Web services description language (wsdl) version 2.0 part 1: Core language. Technical report, W3C (2007)