Integrating Information Systems Components: A Situation-Driven Approach[1]

Nicolas Arni-Bloch, Jolita Ralyté, Michel Léonard

University of Geneva, CUI, 24 rue General Dufour,
CH-1205 Geneva, Switzerland
{Nicolas.Arni-Bloch, Jolita.Ralyte, Michel.Leonard}@cui.unige.ch

Abstract. Integration of new components into existing information systems (IS) is a challenging problem mainly because of the data sharing. In this paper we propose a situation-driven approach for IS components (IS-COTS) integration into existing IS. We claim that such an approach has to take into account a large number of situations and therefore has to be built by applying situational method engineering principals and defined as a collection of reusable method chunks. The main contribution of this work consists of the metamodel for IS-COTS definition, the specification of the requirements for the proposed approach and the illustration of our approach with three method chunks.

1 Introduction

Enterprise-wide Information Systems (IS) are very often built of several more or less autonomous IS components. Besides, constant changes and evolution of enterprise organisation and business require for new IS components that have to be integrated with the legacy ones. This situation leads to fragmented IS and therefore to the redundancy between different IS components which introduces the need for a permanent data, process and rules consistency validation. The common part of different IS components represents the interoperability area of the enterprise IS. If no integration is done between these IS components, the interoperability challenge is left to the human using this IS, i.e. the human have to validate "by hand" the consistency between different IS components. Such a human intervention generates extra cost and leads to a poor data quality.

To reduce this interoperability cost and to support the extension of existing IS with new components we aim to develop a new approach for IS components, that we call IS-COTS, integration into legacy IS. It is evident that such an interoperability challenge leads to a large range of situations to be considered and resolved. As consequence, we build our approach following situational method engineering [9] principals which focus on formalising methods in terms of collections of reusable method components and reusing these components according to the situation at hand.

[1] This work was partially supported by the Swiss National Science Foundation. N° 200021-103826

The main objective of this paper is to define the notion of IS-COTS and to specify the requirements for an approach supporting IS-COTS integration into existing IS.

The remainder of this paper is in five sections. In section 2 we analyse the domain of component-based IS engineering and explore how COTS-based software engineering principles can be adapted to IS engineering. Section 3 provides the metamodel of IS-COTS while section 4 specifies requirements for an approach supporting IS-COTS integration in a situation-driven manner. To illustrate our approach we propose three examples of method chunks in section 5. The paper ends with a review of this work and its future perspectives.

2 Component-Based Engineering of Information Systems

IS engineering methods must continuously consider the new forms of technology to support business activities. Introduction of Enterprise Resource Planning (ERP) systems modify deeply IS engineering in the enterprises; even if not all IS domains are covered by ERPs. But, despite the increase of flexibility of such systems, often the organization of the enterprise and/or its business practices have to be considerably transformed to be adequate with the selected ERP. Sometimes this transformation fails. Nevertheless, the ERP approach has success and the main reason of that is the integrated approach for all the data that the ERP system manages.

Another approach of component-based IS engineering is based on the use of generic components, generally called Component off-the-shelf (COTS). The slogan of this approach is “not to reinvent the wheel”. The developers of software and/or information systems are invited to go to the hyper market of COTS (of course it is a virtual one as Capterra [2], CXP [4], KnowledgeStorm [8], ComponentSource [3], etc.), to select the appropriate products, and finally to integrate them into systems. Of course, the integration of COTS into systems requires some glue, mediators, exchanges of messages, controls, adaptors, translators, wrapper, etc. Generally speaking, a COTS is like a black box: the internal part of a COTS is considered as irrelevant during its selection and integration process. Therefore, developers working with COTS know their interface and their functionalities but nothing relative to their internal parts. These COTS are intensively used in software engineering but less in IS engineering. Any system developed with COTS embeds a kind of flexibility; it is relatively easy to replace one COTS by another if the first one is not satisfactory or become obsolete. But in the case of IS development, a COTS can manipulate data which is also part of the IS and which can be shared by various applications. In order to ensure the integrity and the coherence of this shared data, the architecture of the IS should forbid its storage as an internal part of a COTS and should support data integration.

Therefore, we claim that COTS for information systems, that we call IS-COTS, should be *white boxes*. Due to the fact that IS-COTS share data previously stored in the IS when they are integrated and executed in this IS, the process of integration of IS-COTS into an IS is more sophisticated than the process of integrating traditional software COTS. It must take into account the fact that the IS-COTS and the IS have an overlap of data and this overlap must be overcome. Besides, it is not only a

technical question but also a question of business rules, sharing of responsibilities and transformation of schemata at the static and the dynamic level. Insertion of IS-COTS into a legacy IS is in fact an extension of the IS application domain. Moreover, this extension is at the origin of a lot of new situations relative to the sharing of objects of classes, compatibility between constraints and coordination between transactions but also the emergence of new human roles, new human activities, new business rules and new levels of human coordination.

We claim that IS engineering based on IS-COTS integration can combine the real strength of the ERP approach relative to the integration of data and the real strength of the COTS approach relative to flexibility. Definition of an IS-COTS as a *white box* allow us to clearly identify the overlap between the IS-COTS and the IS under consideration. This overlap specification is necessary to evaluate the impact of the IS-COTS integration and to identify elements that serve as a basis in the integration process. In the next section we provide the definition and the metamodel of an IS-COTS. This metamodel is as a basis for the overlap specification and represents the product part of the methodology for IS-COTS integration.

3 Information System Component – IS-COTS

Traditional object-oriented methods propose two categories of models, static and dynamic, to specify software and information systems. Static models deal with data structure and system architecture definition while dynamic models define system functionalities, activities, states and behaviour. In our opinion, these two modelling perspectives are not sufficient to completely specify an IS. We identify a third perspective, which is specific to the IS engineering, the specification of rules governing the IS and ensuring the integrity of its data. Therefore, from the conceptual point of view the specification of an IS is a triplet <static space, dynamic space, rules space> [16]. The static space represents the structure of the information, the dynamic space captures the manipulations that the information can undergo and finally the rules space represents the constraints that the static space must satisfy.

As defined in [16] an Information System Component (IS-COTS) is a particular IS and can be considered through the same three spaces. In the following we present the metamodel and an example of an IS-COTS.

3.1 Metamodel of an IS-COTS

Fig. 1 represents the metamodel of an IS-COTS illustrating the three spaces: static, dynamic and rules.

Static space (SS). We use an object-oriented approach to represents the structure of the information. As shown in Fig. 1, the main concepts of this space are well known: class, attribute, key, specialization/generalization relationship, and method. Relationships between classes are limited to existential dependencies and are captured in the attributes. To be able to better represent the domain of the IS and to support its evolution, we extend the object-oriented approach with the notion of a *Hyperclass*. “A

Hyperclass is a large class, composed of a subset of conceptual classes of the IS schema, forming a unit with a precise semantic" [15]. Generally speaking, a hyperclass is associated to a specific function inside the IS and each IS-COTS is built on one hyperclass.

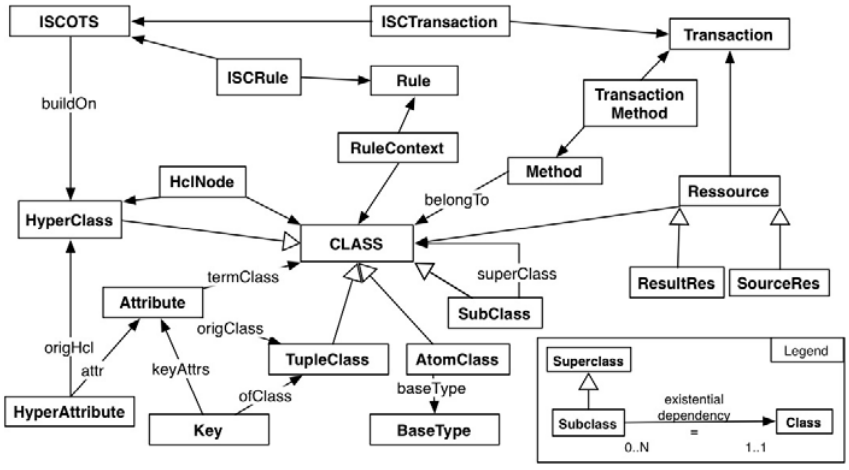


Fig. 1. Simplified metamodel of an IS-COTS

Dynamic space (DS). There are several ways to express the dynamic specifications of an IS such as state-diagrams, object life cycles described with state charts (UML), Petri nets or bipartite nets. We use a bipartite net where one type of nodes represents classes and the other type represents transactions. A transaction is a complete cycle of data processing which changes the state of one or more objects. It is a sequence of operations considered as a single unit. These operations can be the predefined elementary ones such as *Create*, *Retrieve*, *Update* and *Delete* or the invocations of class methods (Turki2003).

Rules Space (RS). The objective of this space is to preserve the coherence, correctness and consistency of the IS during its exploitation. The main type of rules to be considered here is the integrity rules [16] the role of which is to ensure the integrity of the IS data. Besides, business rules are also included in the RS space.

3.2 Example of an IS-COTS

To illustrate the notion of an IS-COTS and its integration into an existing IS, we use a simplified version of the IS of our University, the part which is in charge of courses and students management. This IS was designed to support courses and students related to the Bachelor diploma (in the terms of the Bologna declaration). The static space of this IS is shown in Fig. 2 (a). It provides support to describe and organize

Bachelor courses. It allows to allocate students to the courses corresponding to the followed Bachelor and to store their examination results, to associate teachers to their courses, and to generate several documents like examination reports, Bachelor diplomas, etc. Each person (student or teacher) belongs to an institution that can be either a University or a department of a University or another kind of organisation (e.g. a teacher from an industrial company).

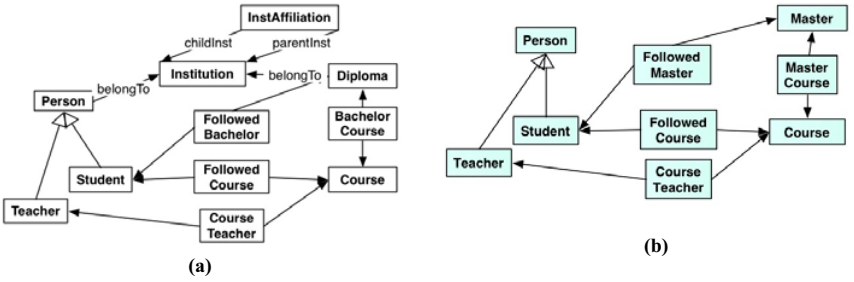


Fig. 2. Static spaces of (a) the IS for Bachelor Diploma management and (b) the IS-COTS for Master Diploma management, (simplified versions)

In order to support the management of the Master diploma in the same manner as the Bachelor diploma, the initial IS has to be extended with a new IS-COTS. The static space of this IS-COTS is shown in Fig. 2 (b). We can already see from this figure that the overlap between these two static spaces is quite important and their integration requires a well guided methodological support. In the next section we specify requirements for a method supporting IS-COTS integration into an IS.

4 Situational Approach for Integrating IS-COTS

According to several authors [6, 10, 11], Method Engineering (ME) is organized in three main phases: method requirements specification, method design and method construction and implementation. We follow this way of thinking to define our approach for IS-COTS integration. In addition to the traditional metamodelling technique, we apply situational method engineering principals such as *modularity*, *reusability* and *flexibility*. The principle of reusability invites to reuse parts of existing methods in the construction of new ones while the principle of modularity provides means to define these parts of methods as method engineering building blocks generally called method fragments [1], method components [5, 17] or method chunks [10, 11]. All these ME building blocks have to be cohesive, autonomous and interoperable [10, 17]. Finally, the principle of flexibility deals with situation-specific method adaptation and/or construction 'on the fly'. According to the situational method spectrum proposed by Harmsen et al. in [7], modular methods are the most flexible ones and modular ME is the most flexible technique to construct new methods 'on the fly'.

In our case, the step of method design is based on the notion of reusable method chunk while the step of method construction uses the assembly technique in order to combine method chunks into a situation-driven and flexible process. Instead of providing one universal methodology for IS-COTS integration we propose to define it as a collection of inter-related and reusable method chunks each of them addressing some specific activity in the IS-COTS integration process. Some chunks can be extracted from existing methods [11] other have to be defined ad-hoc or by using other method engineering techniques such as, abstraction, instantiation, adaptation, etc. [13].

The main contribution of this work is the specification of the requirements for a situational method supporting IS-COTS integration and illustration of a few method chunks to be included in this method.

4.1 Requirement specification for IS-COTS integration

Integration of an IS-COTS into an existing IS has to be considered in two levels: conceptual and system level. The former deals with the integration of conceptual models representing data schemas, integrity rules and transactions while the later aims to ensure compliance with existing data and applications. In this paper we present the part of our approach dealing with the integration at the conceptual level.

Supposing that the required IS-COTS have been selected, we consider that the process of its integration into the IS should be based on five main steps:

1. The adaptation of the selected IS-COTS,
2. The identification of the overlap between the IS-COTS and the IS,
3. The unification of overlap situations if necessary,
4. The construction of the integrated specifications by applying the appropriate integration operators on the identified overlap elements, and
5. The consolidation of the obtained specifications.

During the first step the selected IS-COTS have to be adapted to the system requirements. It can be reduced if its scope is too large, for example, it contains some inadequate classes, transactions or rules. Or in the contrary, it can be necessary to extend the component with some additional elements (classes, attributes, rules, etc.) in order to better prepare it for the integration into the IS. Besides, it can undergo different kind of modifications as renaming, restructuring or reattribution of roles.

The second step consists in identifying overlap situations in the specifications of the selected IS-COTS and the considered IS, analysing these situations, identifying elements for the integration (e.g. classes to be merged) and detecting elements that need to be unified before their integration.

During the third step the overlap situations requiring some unification have to be settled by applying the appropriate unification and transformation operators. We distinguish three types of unification: semantic, structural and functional. Semantic unification mainly concerns concepts naming. For example, if the same name is used for different things or, in the contrary, the same thing is named differently in the two schemas, one of the names have to be modified. Structural unifications deal with structural representation of the information. For example, a concept can be

represented by a class in one schema and by an attribute in the other. In this case, the attribute can be transformed into a class. Finally, functional unification mainly appears in the dynamic space of the specifications and concerns the modification of transactions. It is evident that modifications have to be avoided as much as possible in the IS specification and the most of the transformations have to be done on the IS-COTS specification in order to preserve the integrity of the already existing data and running processes.

The fourth step consists in selecting the appropriate integration operator for each overlap situation and to apply it.

Finally, the integrated specification has to be consolidated. The integration of an IS-COTS into an IS can create new situations which didn't exist neither in the initial IS nor in the IS-COTS. The new integrated system is more than simple addition of two applications. It can provide additional information or functionalities and impose additional rules that have sense only in the new integrated system. Therefore, it can be necessary to add new integrity rules or new transactions in order to guarantee the completeness and the coherence of the integrated specification.

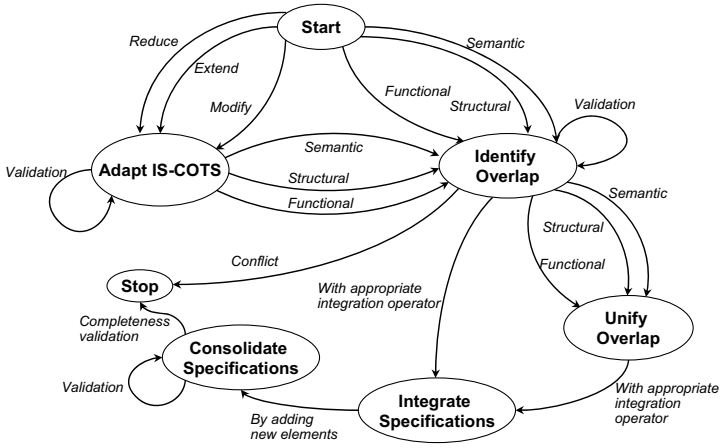


Fig. 3. Requirements Map for the approach supporting IS-COTS integration

Based on this reasoning, we specify requirements for the approach supporting IS-COTS integration as shown in Fig. 3. As advised in [10, 12], we use a strategic process modelling formalism called a MAP [14] to represent the requirement for our approach. MAP provides a representation system based on a non-deterministic ordering of *intentions* and *strategies* to achieve the intentions. It is a labelled directed graph where intentions are nodes and strategies are edges between intentions. Since, many strategies can be used for achieving an intention, MAP allows to represent complex, flexible and situational process models including multiple techniques to achieve the intentions.

Based on the discussion above, the requirements map (Fig. 3) for IS-COTS integration contains five main intentions namely *Adapt IS-COTS*, *Identify overlap*,

Unify overlap, *Integrate specifications* and *Consolidate specifications* and foresee several strategies to achieve each intention. The process model for IS-COTS integration should provide one or several method chunks for each section (a triplet <*source intention, target intention, strategy*>) of this map. The method design step consists in defining these method chunks, each of them being more or less complex, atomic or compound, depending on the complexity of the guideline it have to provide. The metamodel of method chunk was already presented in [10, 11]. In the next section we propose three examples of method chunks to be included in the situational method for IS-COTS integration into IS.

5 Examples of Method Chunks for IS-COTS Integration

In this section we propose three examples of method chunks: the first is related to the overlap identification between the IS-COTS and the IS specifications, the second deals with the integration of specifications and the third one helps to consolidate integrated specifications.

Identify Overlap. The overlap between the IS and the IS-COTS specifications have to be considered in the three spaces: static, dynamic and rules. The method chunk presented in Table 1 helps to identify and describe the semantic overlap in the static space. This identification leads to an *overlap report* that describes all semantic relationships between the IS and the IS-COTS static space specifications. For each couple of classes having similar semantics in the IS and the IS-COTS specification a relationship descriptor is provided.

Integrate Specifications. Based on the overlap report several integration situations can be identified. We propose a method chunk (see Table 2) that aims to realize one of them: the integration of two classes where their objects are in the situation of intersection. The guideline of this chunk specifies operations that must be realized in order to fulfil the goal of this chunk. For some of them the notion of common attributes, methods, transactions, and integrity rules are used. The identification of such common elements is the role of method chunks supporting the *Identify Overlap* phase but which are not presented in this paper due to the lack of space. Another notion used in the guideline of this method chunk is the *Product Requirements*. This notion is used to specify some additional requirements on the product part. For example, in this chunk some evolution primitives must be enabled in order to realize the guideline operations. These requirements also play a role during method chunks selection process. Indeed, if the IS is not able to support these evolution primitives, this method chunk cannot be selected.

Consolidate Integrated Specifications. The method chunk presented in Table 3 helps to consolidate the integrated specifications. It is mainly helpful after the application of the method chunk presented in the previous section. Indeed, the integration of two classes via a common superclass generates a new situation: the integrated specifications have to support object transfer from one subclass to another and requires for new integrity rules and new transactions supporting such a transfer.

Table 1. Method chunk supporting semantic overlap identification and description between similar classes of the IS and IS-COTS

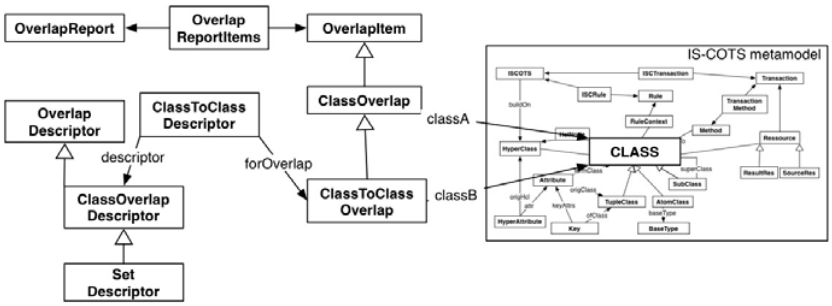
Chunk ID: MC01	**Name:** Semantic Overlap of Similar Classes
Objective: To identify and specify the semantic overlap between similar classes	
Interface: **Situation:** Static space specifications of the IS and the IS-COTS **Intention:** To produce an overlap report that specifies the semantic relationships between the classes of the IS specification and the similar classes of the IS-COTS specification **Body:** **Product Part:** The IS-COTS metamodel (Fig. 1). The overlap report metamodel (figure below).  **Guideline:** For each classe from the IS specification that have a semantic relationship with a class of the IS-COTS, to create an overlap item in the overlap report and to define the relationship descriptor. There are five types of descriptor. Each of them is based on the set of objects (**Cl_{is}) of the IS class Cl_{is} and on the set of potential objects (**$Cl_{is\text{-}cots}$) (objects that will be stored after the integration) of the IS-COTS class $Cl_{is\text{-}cots}$. 1. **Cl_{is} = **$Cl_{is\text{-}cots}$: all objects of the IS class are included in the IS-COTS class and vice versa. 2. **Cl_{is} ⊂ **$Cl_{is\text{-}cots}$: all objects of the IS class are included in the IS-COTS class. 3. **Cl_{is} ⊃ **$Cl_{is\text{-}cot}$: all objects of the IS-COTS class are included in the IS class. 4. **Cl_{is} ∩ **$Cl_{is\text{-}cots}$ ≠ ∅ : some objects are common to the IS and the IS-COTS class and some are not. 5. **Cl_{is} ∩ **$Cl_{is\text{-}cots}$ = ∅ : the IS and the IS-COTS classes have no common objects.	
Application Example: Let us take the example introduced in section 3.2. The overlap between the University IS and the IS-COTS for Master diploma can be analysed as follow: • **IS.Person = **IS-COTS.Person • **IS.Teacher = **IS-COTS.Teacher • **IS.Student ∩ **IS-COTS.Student ≠ ∅ : Students of the IS are at the Bachelor level while those of the IS-COTS are at the Master level. Therefore, there are three categories of students: (1) those that are only known as Bachelor students, (2) those that have not done their Bachelor in this University and are only known as Master students and finally (3) those that have finished their Bachelor in this University and now are Master students. • **IS.Course ∩ **IS-COTS.Course = ∅ : Courses from the IS represent Bachelor courses while those from the IS-COTS represent Master courses. As a course can be only a Bachelor or a Master course, the intersection of the objects is empty. • **IS.Diploma ∩ **IS-COTS.Diploma = ∅ : Diplomas from the IS represent Bachelor diplomas, those from the IS-COTS represent Master diplomas. As a diploma can be only a Bachelor or a Master one, the intersection of the objects is empty.	

Table 2. Method chunk supporting the integration of two classes via generalization operation

Chunk ID: MC02	**Name:** Integration of two classes by creating a common super class

Objective: To integrate two classes via generalization operation

Interface:

Situation: Class *Clis* from the IS specification and class *Clis-cots* from the IS-COTS specification

Intention: To integrate two classes by creating a third class that is a generalization of the two initial ones

Body:

Product Part: The IS-COTS metamodel (Fig. 1).

Product requirements (P.Req.):

addSuperClass: The integrity of the objects must be guaranteed when a superclass is added to a class.
moveAttributeToSuperClass: The integrity of the objects must be guaranteed when moving attributes form a subclass to its superclass.
moveMethodToSuperClass: It is possible to move a method from a subclass to its superclass.
moveTransactionToSuperClass: It is possible to move a transaction from a subclass to its superclass.
moveIRToSuperClass: It is possible to move an IR from a subclass to its superclass.

Guideline:

In the IS:

1. Add a common class Cl' as a generalization of Clis and Clis-cots.
2. Change the IS spec: to make Cl' a superclass of Clsi (P.Req. addSuperClass).
3. Move all attributes that are common with Clis-cots from Clis to Cl' (P.Req. moveAttributeToSuperClass).
4. Move all methods that are common with Clis-cots from Clis to Cl' (P.Req moveMethodToSuperClass).
5. Move all transactions that are common with Clis-cots from Clis to Cl' (P.Req. moveTransactionToSuperClass).
6. Move all integrity rules that are common with Clis-cots from Clis to Cl' (P.Req. moveIRToSuperClass).

In the IS-COTS:

1. Delete from the Clis-cots all attributes that are common with Clis.
2. Delete from the Clis-cots all methods that are common with Clis.
3. Delete from the Clis-cots all transactions that are common with Clis.
4. Delete from the Clis-cots all integrity rules that are common with Clis.
5. Change the IS-COTS spec: to make Cl' a superclass of Clis.

Application Example: Figure below shows the integrated static space of the University IS and the Master IS-COTS.

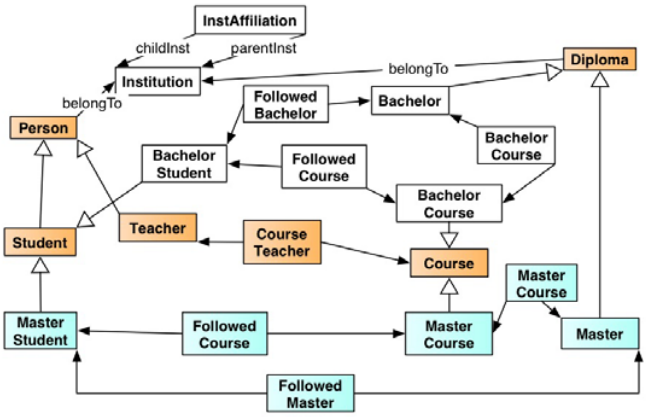
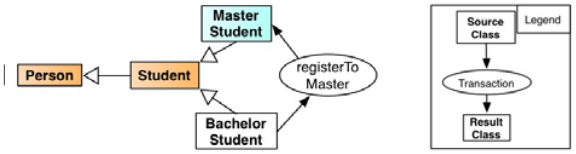


Table 3. Method chunk supporting the consolidation of the integrated specification

Chunk ID: MC03	**Name:** Consolidate integration made via superclass creation
Objective: To consolidate the integrated specification that has been obtained by creating a common superclass.	
Interface: **Situation:** Class Clis from the IS specification, class Clis-cots from the IS-COTS specification and class Cl' - common super class to Clis and Clis-cots **Intention:** To consolidate the integrated specification by adding new transactions and integrity rules **Body:** **Product Part:** The IS-COTS metamodel (Fig. 1). The overlap report model (see chunk MC01). **Guideline:** If the relation descriptor between integrated classes is $**Cl_{is} \cap **Cl_{is\text{-}cots} = \varnothing$ then add a consistency rule ensuring that each object can be only an object of Clis or of Clis-cots. If the relation descriptor between integrated classes is $**Cl_{is} \cap **Cl_{is\text{-}cots} \neq \varnothing$ then determine if a new rule is needed between Clis and Clis-cots and add one or more transactions for objects transfer from one class to another.	
Application Example:  In our example, the integrated IS has to manage two kinds of student: Bachelor and Master. As defined in the overlap report, a student object can belong to both the BachelorStudent class and the MasterStudent class. Therefore, we need to add a new transaction allowing a Bachelor student to become a Master student. Besides, a new rule has to be added in order to ensure that a Bachelor student has finished his Bachelor degree before beginning Master studies. This rule is expressed as follows: ON ENTER in MasterStudent : obj exist in BachelorStudent AND obj.hasBachelor == True where obj represent a student.	

5 Conclusion

While the use of IS is growing in all areas of our society, the diversity of their application domains increases as well. To address this diversity, ERP systems and Components of-the-shelf, also named COTS, are proposed as building blocks to compose new IS and to evolve the existing ones. In this paper we claim that, in the contrary to the software engineering, COTS for IS engineering cannot be proposed as *black boxes* and a simple plug-in process is not sufficient to integrate COTS into existing IS due to the data, processes and rules overlap. Therefore, in the domain of IS engineering we need a new kind of COTS provided as *white boxes* as well as we need a new kind of approaches to deal with the complexity of their integration. In this perspective we see the following contributions of this paper:

- The notion of IS component that we call IS-COTS and its metamodel.
- The requirements specification for a situation-driven approach supporting IS-COTS integration into existing IS. The MAP formalism used for requirements representation as a strategic process model enables their evolution. Indeed, it is quite simple to add new strategies and intentions into the requirements map.

– The examples of method chunks to be integrated into our approach. We claim that such an approach should be based on situational method engineering principals, i.e. it has to be modular, reusable and flexible. Therefore, we aim to define this approach as a collection of inter-related method chunks dealing with a huge number of IS-COTS integration situations. This kind of flexibility and adaptability to a specific situation lead us to a new kind of approach for IS engineering.

Currently we focus our effort on identifying and evaluating different situations that can occur in the IS-COTS integration process and defining method chunks satisfying these situations. A tool support is also under development.

References

1. Brinkkemper S, Saeki, M. and Harmsen, F. (1999) Meta-Modelling Based Assembly Techniques for Situational Method Engineering, *Information Systems,* **24**(3), pp. 209–228.
2. CAPTERA: http://www.capterra.com/, last visit 23 February 2006.
3. Componentsource: http://www.componentsource.com/, last visit 23 February 2006.
4. CXP: http://www.cxp.fr/cxp/, last visit 23 February 2006.
5. Firesmith, D.G. and Henderson-Sellers, B., 2002, *The OPEN Process Framework – An Introduction*. Addison-Wesley, Harlow, UK.
6. Gupta, D. and Prakash, N. (2001) Engineering Methods from Method Requirements Specifications, *Requirements Engineering,* **6**(3), pp. 135–160.
7. Harmsen, A.F., Brinkkemper, S. and Oei, H. (1994) Situational Method Engineering for Information System Projects. In Olle T. W. and A. A. Verrijn Stuart (Eds.), *Methods and Associated Tools for the Information Systems Life Cycle,* North-Holland, pp. 169-194.
8. KnowledgeStorm: http://www.knowledgestorm.com/ , last visit 23 February 2006
9. Kumar, K. and Wellke, R. J. (1992) Methodology Engineering: A Proposal for Situation Specific Methodology Construction, In *Challenges and Strategies for Research in Systems Development,* (Eds, Cotterman W.W. and Senn J.A.), John Wiley & Sons, pp. 257–269.
10. Mirbel, I. and Ralyté, J. (2006) Situational Method Engineering: Combining Assembly-Based and Roadmap-Driven Approaches, *Requirements Engineering,* **11**(1), pp. 58–78.
11. Ralyté, J. and Rolland, C. (2001) An Approach for Method Reengineering. *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER2001)*, Springer-Verlag, LNCS 2224, pp.471-484.
12. Ralyté, J. and Rolland, C. (2001) An Assembly Process Model for Method Engineering. *Proceedings of the 13th Conference on Advanced Information Systems Engineering (CAISE'01),*Springer-Verlag, LNCS 2068, pp. 267-283.
13. Ralyté, J., Rolland, C. and Deneckère, R. (2004). Towards a Meta-Tool for Change-Centric Method Engineering: a Typology of Generic Operators. *Proc. of the 16th Conf. on Advanced Information Systems Engineering (CAISE'04),* LNCS 3084, Springer-Verlag, pp.202-218.
14. Rolland, C., Prakash, N. and Benjamen, A. (1999) A Multi-Model View of Process Modelling, *Requirements Engineering,* **4**(4), pp. 169–187.
15. Turki, S. and Léonard, M. (2002) Hyperclasses: towards a new kind of independence of the methods from the schema. *Proceedings of the 4th Int. Conference on Enterprise Information Systems,* ICEIS'2002, Vol.2, pp. 788-794, ISBN: 972-98050-6-7. Ciudad Real, Spain.
16. Turki, S., Léonard, M. and Arni-Bloch, N. (2003) From Hyperclasses to IS Components. *Proc. of the 10th Int. Conference on Concurrent Engineering* (CE'2003), Madeira, Portugal. R. Jardim-Goncalves, H. Cha, A. Steiger-Garcao (eds.), Balkema Publishers. pp. 235-242.
17. Wistrand, K., and Karlsson, F. (2004) Method Components – Rationale Revealed. *Proc. of the 16th International Conference on Advanced Information Systems Engineering* (CAiSE'04), Riga, Latvia, Springer LNCS 3084, pp. 189-204.