Information System Services Generation of Business Services Specification and Based on a System-of-Services Logical Architecture Pattern

Jacques Simonin, Pierre-Yves Pillain, Didier Gueriot, Johanne Vincent

International Journal of Cooperative Information Systems

World Scientific
www.worldscientific.com

INFORMATION SYSTEM SERVICES GENERATION FROM BUSINESS SERVICES SPECIFICATION AND BASED ON A SYSTEM-OF-SERVICES LOGICAL ARCHITECTURE PATTERN

JACQUES SIMONIN

IMT Atlantique, Lab-STICC, UMR CNRS 6285, Avenue du Technopôle, Brest, 29238, France
jacques.simonin@imt-atlantique.fr

PIERRE-YVES PILLAIN

Université de Bretagne Occidentale, Lab-STICC, UMR CNRS 6285, 6 Avenue Victor Le Gorgeu, Brest, 29238, France
pierre-yves.pillain@univ-brest.fr

DIDIER GUÉRIOT

IMT Atlantique, Lab-STICC, UMR CNRS 6285, Avenue du Technopôle, Brest, 29238, France
didier.gueriot@imt-atlantique.fr

JOHANNE VINCENT

IMT Atlantique, Lab-STICC, UMR CNRS 6285, Avenue du Technopôle, Brest, 29238, France
johanne.vincent@imt-atlantique.fr

Received Day Month Day
Revised Day Month Day

The generation and design of the service architecture of an information system is complex. It depends more on the vision of the service than on the vision of the service inside an information system. An information system is indeed a system of services that can contain thousands of services. The lack of consideration of constraints imposed by the information system makes it difficult to reuse these services. Another strong constraint is that an information system service must support a business service. The proposed approach allows information system services to be generated in accordance with the business services specification and their logical architecture to be automatically designed by respecting a logical architecture pattern of the system-of-services. An information system services generation algorithm allows being consistent with the logical architecture pattern during this generation. The definition of coherence and coupling properties makes it possible to evaluate the relevancy of the system-of-services. A use case shows the value of these properties in making the logical architecture of the service system more relevant to business services.

Keywords: Business service; System-of-services; Information system service; Model transformation.

1. Introduction

The complexity of an enterprise's Information System (IS) is both quantitative since several hundred or even several thousand applications coexist there, but also qualitative

with the need for flexibility linked to the evolution of the company's strategy, and for agility that offers applications that meet users' needs. Almost twenty years ago, the service paradigm associated with an application that offered an approach to reduce the complexity of an IS appeared.[1]

This paradigm is extended with service-oriented architecture (SOA), which provides a cohesive link between the enterprise's business and the IS.[2] Services that provide an external view of IS applications, or IS services, are orchestrated to deliver business services, i.e. services that specify the company's business processes. This orchestration is made possible by a bus of services allowing access to the services offered by the various IS applications.

The governance of these different IS applications is in most significant companies managed within an Enterprise Architecture (EA) framework.[3] The alignment between business services and supporting IS services is relevant in an EA approach, as they improve the IS governance.[4] The definition of the services to be used in an IS must be consistent with the company's strategy, which is reformulated, in part, with the business services offered to customers. For example, if a commercial enterprise's strategy targets the customer according to its market (enterprise or individual, for example), then an ordering service provided by the IS should target a specific market (a service specific to a company and a service specific to an individual, in the previous example).

Interactions, whether at the business level (collaboration between the sales department and the delivery department), or at the application level (use of data produced by an ordering application by a delivery application), are based on this service concept. The dynamic aspect associated with these services thus appears in all EA frameworks. The use of a business service is dynamically represented during the course of a company's business process. Because of the alignment between the business and the IS,[5] a dynamic representation of the IS services supporting a business service is also relevant. For example, the ArchiMate language, which is one of the basic languages associated with EA, highlights service concepts and the relationships between them.[6] ArchiMate first uses the concept of organizational service to represent the external view of business behavior, for example, the service of a vendor taking an order for a telecommunications product. Then, the realization of this organizational service by IS application is carried out using the application service concept, which is the external view of an application. In our example, it is a service providing the ordering of a telecommunications product for a business customer assigned to the interface of an enterprise market control application (API: Application Programming Interface). This application service requires for its deployment an infrastructure service defining the useful technologies. For example, the application service can be deployed on JEE (Java Enterprise Edition) infrastructure services.

The EA's TOGAF (The Open Group Architecture Framework) framework,[7] with the method it proposes to design the EA allows a good integration of the business during the IS architecture.[8] This ADM (Architecture Development Method) is the major choice criterion of a large majority of companies for a single or hybrid EA's framework.[9] That is

the reason why the meta-modeling used to automate the IS service development are mostly chosen from TOGAF. Meta-modeling is the representation of the concepts useful, here, for IS service development, and their associations (for example, the concept of "IS service" associated with the concept of "platform service", because the former is deployed on the latter). Among the concepts composing these meta-models, two of them are at the heart of our contribution which targets the generation of IS services from the specification of business services: (1) the business service defined in the TOGAF business architecture meta-model and (2) the IS service in the TOGAF application architecture meta-model.

Data concepts make it possible to raise the issue of data quality and integrity during its life cycle, for example for the telecommunications field[10] chosen for illustration in this introduction. This alignment problem between a data entity (business architecture) and a physical data (application architecture) is one aspect of the misalignment between business and IS.[11] This misalignment between business and IS leads to a break between the simultaneous consideration of strategy in the business and in the IS.[12] The implemented view of this IS, called IT (Information Technology), when out of order, even temporarily, is penalized by the development costs incurred by the evolutions necessary to bring it into compliance with the business. The logical enterprise architecture is interesting in this context of rupture because it is the result of a broad vision of the company that reconciles the company's decision-makers and those who ensure IT governance.

However, the logical architecture of an IS is complex because of the abstract nature of these components, grouping functions, and of the relationships between these components. We have seen the effects of this complexity on 2nd year Master students (at University and engineering school) during courses focusing on the architecture of a software application, based on services, in the context of EA. EA was proposed to them as a constraint for their development, either by a logical architecture pattern of the IS in which the software was included, or by a J2E development environment. What was clearly missing to them was the justification of the proposed IS logical architecture pattern, and the possibility of automating, from all the requirements, the design of the logical architecture, then the physical architecture and finally the coding of IS services. This is the reason why we propose, in the first part of the contribution, a pattern establishing these relations from a typing of the logical components of the IS. From this logical architecture, the second part of the contribution of the paper is an algorithm resulting in the generation of information system services supporting a business service. This generation is proposed here in a context of system of information system services characterizing SOA approach. In order to automatize this generation, a logical architecture of the system of information system services is needed first. Our contribution targets the transformation resulting in a logical architecture of an IS service, as a result of the algorithm, which comes before the usual transformation resulting in a physical architecture of an IS service. Each attribute of a given entity produced by a business service can thus be contextually associated with a logical application component. The

generation algorithm is based on this contextual association allowing the design of information system services supporting a business service. The automation of this design is proposed within the framework of an MDA (Model Driven Architecture) approach. The automated generation of an information system service means the implementation of model transformations from the business service analysis to the code via the logical architecture and the resulting applicative architecture. This approach should enable to simplify the generation of services,[13] which is the basis of the orchestration of these services in order to implement a business process. Moreover, the definition of the coherence and coupling properties of information system services in relation to the business services, or of logical application component model in relation to information system services, which we propose, allows us to evaluate the relevance of a system-of-services. The implications, of improving the evaluated measure of these properties, on the business services, on the generated information system services or on the logical application components, are particularly focused in the use case.

The paper is organized as follows. Background and related work regarding system-of-services, SOA, and service design are described in Section 2. The concept of packaging system, such a system-of-services, is defined and improved in the Section 3 by a proposal of pattern enabling to design the architecture of a system-of-services. The usefulness of this pattern to generate services of an IS from a business service is underlined by a generation algorithm in Section 4. A use case is presented in Section 5, applied to a management IS. In Section 6, assessments resulting from this example are described, as well as risks for using such approach. Conclusions and perspectives are summarized in Section 7.

2. Background and Related Work

First, this section examines previous works on the alignment of IS viewpoints with business viewpoint, as defined by EA, including SOA approach, and on abstract level architecture for integration of a service into a system-of-services. Then, the section presents related work about the generation of IS services from business services.

2.1. *Enterprise Architecture and system-of-services*

A system-of-services is often described in companies by a catalogue of services offered to IS users in order to carry out their activities with the help of IS applications. This catalogue allows IS application developers to reuse existing services that provide them with the data their application needs. The orchestration of such IS services, in order to support a business service, highlights the relationships and interactions between the IS services and justifies the term of system.

Alignment between the business and IT is a problem in the industry where collaboration between business experts and IT designers is complex because of specific contexts.[14] This complexity is induced by different points of view about the company.[15] SOA with its properties of flexibility and agility allows the business to design IT services with IT partners.[16] These services are thus more explicit because they are designed and

orchestrated directly from the business point of view.[17] The concept of service can thus be applied to this business point of view in order to define the business service as expected by the business while the IT service will result from this business service taking into account IT-specific constraints.[18] Service oriented architecture can then be extended to IS design with the concept of IS service, which highlights the functional aspect of the service compared to the IT service.[19] This abstract level of a system-of-services is the logical one, which maps with the system viewpoint of Zachman's framework.[20] This functional aspect is integrated in the logical architecture for EA's frameworks as TOGAF, more precisely in the application architecture part of this framework.[7]

As pointed out in the introduction, there are two concepts of TOGAF which are chosen to frame the contribution of this paper:

- For business architecture, the concept of a business service associated with one or more business processes (for example, the three business services of reading the catalogue of telecommunications products, creating a business customer and taking orders for a telecommunications product that support an order process).
- For application architecture, the concept of IS service associated with the automation of a business service (for example, an IS service for consulting a catalogue of telecommunications products offered by the application managing the computerized labels and prices of products).

These concepts compose a part of the TOGAF meta-model of the Services Extension. They are completed in Fig. 1 by the logical application component concept, which specifies the application component concept defined initially in this extension. The logical application component is another concept of the TOGAF meta-model useful here for the IS service logical architecture.

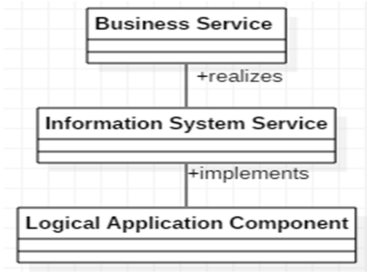

Fig. 1. TOGAF Services Extension meta-model.

For data architecture as defined by TOGAF, the concept of service does not exist, but it is strongly linked to business architecture or application architecture. The concept of data entity is indeed associated (provided or used) with the concept of business service, as well as that of physical data associated with the IS service that provides or uses this physical data (the physical data defining a product with its wording and price is for example provided by a service for consulting a catalogue of telecommunications products).

The synergy between the system-of-services architecture as defined by SOA and EA is nevertheless limited to the recommendation of specific technologies.[21] Our

contribution proposes to extend this synergy to the logical architecture of a system-of-services and its counterpart in EA, for example the application architecture in TOGAF.

The service concept is considered by the SOC (Service-Oriented Computing) approach as central in the development of IS applications.[22] From this approach, the usefulness of model-driven development becomes interesting in order to ensure the integrability and interoperability of IS services. The use of MDA approach for SOA is first and foremost classical in order to integrate SOA technical solutions, resulting from non-functional requirements, during service development.[23] The design of services addresses isolated IS services from one business service instead of an exhaustive design of IS services from an orchestration of business services. However, orchestration can be handled in some frameworks, such as ArchiMeDes.[24] Thanks to rules involving a PDM (Platform Dependent Model) specific to SOA, this framework transforms an orchestration of a logical service model viewed as a PIM (PIM: Platform Independent Model), into a PSM (PSM: Platform Specific Model). The orchestration of business services is supported here by an orchestration of IS services. The completeness results from the algorithm of generation constrained by a context that is the system-of-services. Taking into account a context in the MDA approach[25] is interesting because it would allow the automaticity of the generation proposed in our contribution.

2.2. *Generation of IS services from business services*

The architecture defining how IS services must be developed in order to support business services introduce the concept of system-of-services. The objective of this architecture is the integration of an IS service into the system-of-services of the company.[26] In this system-of-services, the integration of an IS service can be done by discovery (in order to reuse it)[27] or development (in order to support a business service) respecting the property of low coupling between IS services.[28] The architecture of the IS services can be integrated in EA in accordance with an ontology targeting integration.[29] This integration is also being studied from a systemic point of view (a system is composed of units that can be linked by static relationships allowing them to interact dynamically) with the concept of system-of-systems. An example is the integration, of computer applications, which transforms these systems into subsystems composing an IS The IS thus is a system-of-systems. We find the architecture layers associated with subsystems that make up a system-of-systems. What is interesting is that the lowest layer of the systemic approach, that of the units composing the lower-level subsystem, is not addressed, in particular the coupling property which is on the other hand highlighted for subsystems.[30]

The system-of-systems is also better treated than the system-of-services in terms of its architecture. This is the case, for example, for the integration of a system into a system of systems.[31] An interesting point is that the modeling of an IS should be considered first at the abstract level of a system-of-systems.[32] Unfortunately, this abstract level is not emphasized much in the studies of systems-of-services. The evaluation of system-of-systems architecture is rather achieved by experts of each level and based on a mapping between activity and system.[33] However, it does not benefit of development traceability

between these levels. Traceability is indeed a quality attribute of a system-of-systems [34] and is essential for the life-cycle (development process, maintenance process, and operating process) of a complex system[35] such as a system-of-systems, or a system-of-services. Traceability is generally based on a documentation reproducing the activities of the processes characterizing the processes of the system life cycle. As mentioned above, traceability in SOA focuses on the business and the physical,[36] and therefore does not take into account an abstract intermediate level of the system-of-services.

The model-driven approach for this architecture should assist the evaluation and the traceability of a system-of-systems. More generally, the model-driven approach is essential both for specifying business processes and for developing the IS services that support them.[37] This approach is beneficial in the context of system integration in a system-of-systems with patterns,[38] but this solution suffers from the absence of a global system-of-systems architecture solution.[39,40] Indeed, each model transformation uses one task as input and results into one IS service.

Still in the model-driven approach, transformations of business processes modeled with BPMN (Business Process Model and Notation) into a logical service architecture modeled with SoaML (Service oriented architecture Modeling Language) are implemented with any model transformation language. These transformations consist mainly either of transformations of message, which stipulate data entities exchanges, into service interfaces,[41] or of transformations of task into services.[42] Compared with these two examples of the generation of IS services from business processes, what is missing is the failure to take into account the system-of-services that encapsulates the services of an IS. Indeed, without this constraint for any IS service logical architecture to conform to an enterprise-specific system-of-services logical architecture, it is difficult to promote the reuse of IS services by other IS services. This is the foundation of our contribution to generate IS services from business services in a consistent way across an enterprise's IS. This logical architecture of a system-of-services of a company must therefore be designed in advance of any generation of IS services in this company.

In a similar way to system integration in a system of systems, the integration of a service into a system of services requires the creation of abstract views of the developed services IS.[43] Nevertheless, an abstract global view of the service system is not deeply treated. The global integration difficulty is the granularity of the design of the components at the abstract level. A multi-level approach for the IS service is the most common solution. Each level maps with a granularity of a service, which is associated to a view of this service (process, business, composite…).[44] However, the modularity is often based on environment criterions and can be assisted by an estimation of the technical potential of each module in software design.[45] Microservices designed in order to develop adaptive system-of-services, as cloud system,[46] are considered as independent component containers. The sharing of components is then difficult to apply during the development of the IS services. Modularity is also a fundamental property in the design of heterogeneous systems fusion architecture, which offers solutions to the system of-systems architecture.[47] Functions are grouped together based on the detection of the

sharing of the same information in order to best reuse a function. Nevertheless, there is no a priori generation of these functions merging at the abstract level.[48] The creation of IS services is covered in [49], where an IS service is virtualized from the ability of an agent responsible of the execution of a system's use case. This service engineering does not take into account a priori global design of a system-of-services favorable to the IS services reuse, but the uses of this system.

One observation is that the appropriate transition between business and IS is not conducive to an automated development of IS services. Instead of ontology and hierarchical modeling solutions (i.e. refinement solutions), we prefer to design a system-of-services logic model, as global solution for this system, in which IS services must be integrated, or reused. Indeed, on the one hand, an ontology will specify a business model of data entities based on the expertise of the addressed business. The transformation of such a model into a logical architecture model also requires the consideration of the business services as used in the company. These business services are impacted by the organization of the company. However, this organization specific to a company and an ontology specific to a business domain in which the company is working can be contradictory. On the other hand, in the case of refinement, the problem is the granularity of the logical operation when it is transformed into a physical operation, i.e. ready for implementation. The objective of automating the generation of the logical architecture of IS services is difficult to achieve because of this uncertainty. Another observation is the usefulness of the lowest level of systemic approach to automatically develop these IS services (from IS services generation to IS services coding). The goal is to simplify the interaction of business experts with IS services architecture through the functions describing the system-of-services. For this purpose, measurable properties are proposed below to define a suitable system-of-services.

Related work makes it possible to highlight the specificity of the contribution, where the IS is considered as a constraining context for the generation of IS services from business services. This context must be shared by all generated IS services and must therefore be designed beforehand. The first expectation addressed in section 3 is a solution for modelling IS seen as system-of-services.

3. New Pattern for System-of-Services Design Based on Packaging System Properties

We first define in this section a packaging system whose properties applied to a system-of-services would automatically enable the generation of IS services and their logical architecture in relation to the specification of a business service. This generation is detailed in an algorithm in Section 4 where the design of IS services is guided by the design of the logical architecture of the IS presented below. Then, a pattern is suggested in order to design the logical architecture of a system-of-services[50] satisfying the properties of a packaging system.

3.1. *Packaging system definition*

Properties defining a packaging system and illustrations of the expected properties of a packaging system for three well-known systems, especially a system-of-services, are described in this section.

3.1.1. *Packaging system properties*

Packaging is defined here as in UML (Unified Modeling Language) by a grouping of coherent elements.[51] A system is also defined by a grouping of coherent subsystems. The coherence of a subsystem (differentiated from its cohesion)[52] is conditioned by the requirements specification that the system's subsystems have to solve. A generic definition of coherence is proposed below in order to extend the coherence property to the subsystem's units conditioned by the specification of each subsystem. A requirement that conditions the subsystem design, and a subsystem that conditions the units design, define an expectation in the both following generic definitions. The following definitions of coherence and coupling property are applied to elements of a set SO of solutions that should solve some elements of EX, set of expectations.

Definition of coherence. One element so_1 of SO is ***coherent*** with the set of expectations EX if $\exists ex_1 \in$ EX, such as so_1 solves ex1 and only ex1.

In Fig. 2, only so_1, so_3 and so_5 from SO are coherent with EX.

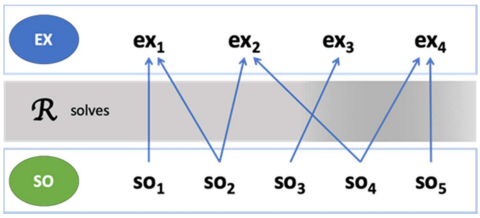

Fig. 2. Example of a solution set SO solving a set EX of expectations. A solution so_k solves (at leastly partly) one or more expectation ex_n. The relation "solves" between EX and SO is denoted $\mathcal{R}$.

An evaluation of the coherence of SO could thus be the ratio of the amount of SO elements coherent with EX, to the SO total amount of elements.[53] Let us therefore consider a MCH ***measure of the coherence*** of a set SO conditioned by a set of expectations EX:

$$MCH(SO, EX) = card(\{so \in SO; so \text{ is coherent with EX}\}) / card(SO) \quad (1)$$

where *card(A)* returns the cardinality of the set A.

The behavior of the MCH measurement is based on the following two axioms:[54] (1) The lower the coherence, the lower its measurement decreases, and therefore, the closer its evaluation is to 0. (2) The higher the coherence, the higher its measurement increases, and therefore, the closer its evaluation is to 1.

In Fig. 2, the measurement of the coherence of the set SO is MCH(SO,EX) = 3 / 5.

If each subsystem is designed from units, the units interacting with each other during an instantiation of a subsystem, and thus having a static relationship between them,[55]

define a coupling property. The following extended definition of coupling property enables to specify also the coupling between the system's subsystems conditioned by the system requirements specification.

Definition of coupling. Two elements so_i and so_j from a set SO of solutions solving of a set EX of expectations are ***coupled*** if they are linked by a static oriented relationship deduced from a dynamic interaction required by an instance of an expectation of EX.

Thus, so_i and so_j are coupled if either

(a) $\exists ex \in EX$ / so_i solves ex and so_j solves ex,

(b) or $\exists so \in SO$ / so and so_i are coupled and so and so_j are coupled.

Then, the ***coupled*** relation defines an equivalence relation on SO, allowing to build $\mathcal{P}(SO)$, the set of its equivalence classes.

In Fig. 2, for instance so_1 and so_2, so_2 and so_4, so_4 and so_5 are coupled through condition (a) involving so_1, so_2, so_4, and so_5 are coupled through condition (b). Only so_3 is not coupled with any other solution. Then, $\mathcal{P}(SO)$ is here a two-elements set $\mathcal{P}(SO) = \{\{so_1, so_2, so_4, so_5\},\{so_3\}\}$.

The evaluation of a coupling is more common.[56] One MCU ***measure of the coupling*** of a set of elements SO solving a set of expectations EX could be:

$$MCU(SO, EX) = 1 / card(\mathcal{P}(SO)) \quad (2)$$

where $\mathcal{P}(SO)$ is the partition of set SO built according to the ***coupled*** equivalence relation existing on SO through the set EX and the "solve" relation between SO and EX.

The behavior of the MCU measurement is also based on two axioms:[54] (1) The lower the coupling, the lower its measurement decreases, and therefore, the closer its evaluation is to 0. (2) The higher the coupling, the higher its measurement increases, and therefore, the closer its evaluation is to 1. Thus, the measure of coupling of the set SO in Fig. 2 is MCU(SO,EX) = 1 / 2.

We define a packaging system from coherence and coupling properties to be verified by the subsystems on one hand and by the units on the other hand. This definition is based on the properties necessary for modularization, which are low coupling between modules and high consistency within each module.[57] A module for a system is a subsystem of this one.

Definition of packaging system. Knowing a set of requirements specifying the system, a ***packaging system*** is a package of subsystems, exhibiting low coupling and high coherence between them. Each subsystem is composed of units with high coupling and low coherence between them, knowing the specification of each subsystem.

The awaited packaging system properties are illustrated below with system-of-services compliant with the SOA approach.

Packaging systems are represented in this paper through UML2[58] class diagrams, where the dependency stereotypes "solves" from SO elements on EX elements matches with the $\mathcal{R}$ solving relationship expressed in the definitions of coherence and coupling.

3.1.2. *Packaging system-of-services*

As shown in Fig. 3, a system-of-services, where a service is an IS service, is a potential packaging system. In relation to the systemic approach, the system is a set of IS services, and an IS service is a subsystem. This system-of-services support the business services triggered by the "customers" of these last ones.[59] Considering the logical architecture of an IS service, the units of a subsystem are the LACs (Logical Application Components as defined in the TOGAF meta-model) required during the instantiation of an IS service.

In Fig. 3, two relations $\mathcal{R}$ exist:

- A "supports" relation, between ISS and BS, considering the case where an IS service solves a business service, for instance ISS1 IS service supports BS1 business service.
- A "is required by" relation, between LAC and ISS, when a LAC is required by an IS service during the logical design, for instance LAC1 and LAC2 LACs are required by the ISS1 IS service.

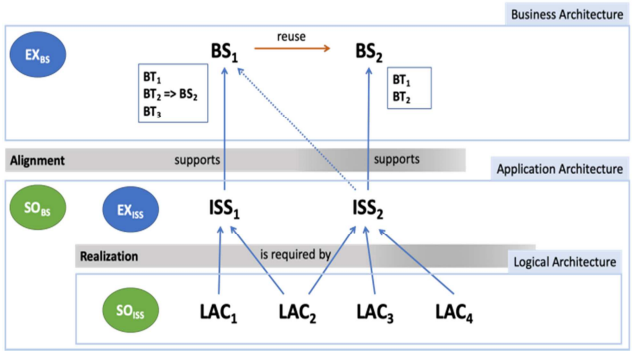

Fig. 3 – Example of a system-of-services.

The subsequent sections propose coherence (MCH) and coupling (MCU) evaluations for several configurations of system-of-services, on both levels: (a) IS services solving business services and (b) logical application components solving IS services.

3.1.2.1. Coherence and coupling evaluations for ISS supporting BS

For the illustration of the IS services coherence (see Eq. 1) and coupling (see Eq. 2) evaluations, two examples are proposed (see Fig. 4). In Example ISS/BS#1, three IS services are designed with one supporting a task common to two business services. In Example ISS/BS#2, each business service is solved by one IS service.

About Example ISS/BSS#1, according to Eq. 1, the coherence MCH is not maximal because the *ISSReadDepartment* IS service supports both business services (a task of reading a department is common to *BSCreateEmployee* and *BSCreateDepartment*); only two over three IS services are coherent with business services set. On the contrary, in Example ISS/BS#2, MCH is maximum because each IS service solves only one expectation

In ISS/BS#1, *BSCreateEmployee* is supported by the orchestration of *ISSReadDepartment* and *ISSCreateEmployee* and *BSCreateDepartment* by the

orchestration of *ISSReadDepartment* and *ISSCreateDepartment*. The partitioning gives then only one subset due to the shared use of *ISSReadDepartment* in both orchestrations. According to Eq. 2, the coupling MCU is then a maximum.

Conversely, ISS/BS#2 shows a lower coupling since there is no orchestration linking the two IS services. The partitioning conditioned by the business services, gives then two complementary subsets of IS services and the coupling measure decreases to 0.5.

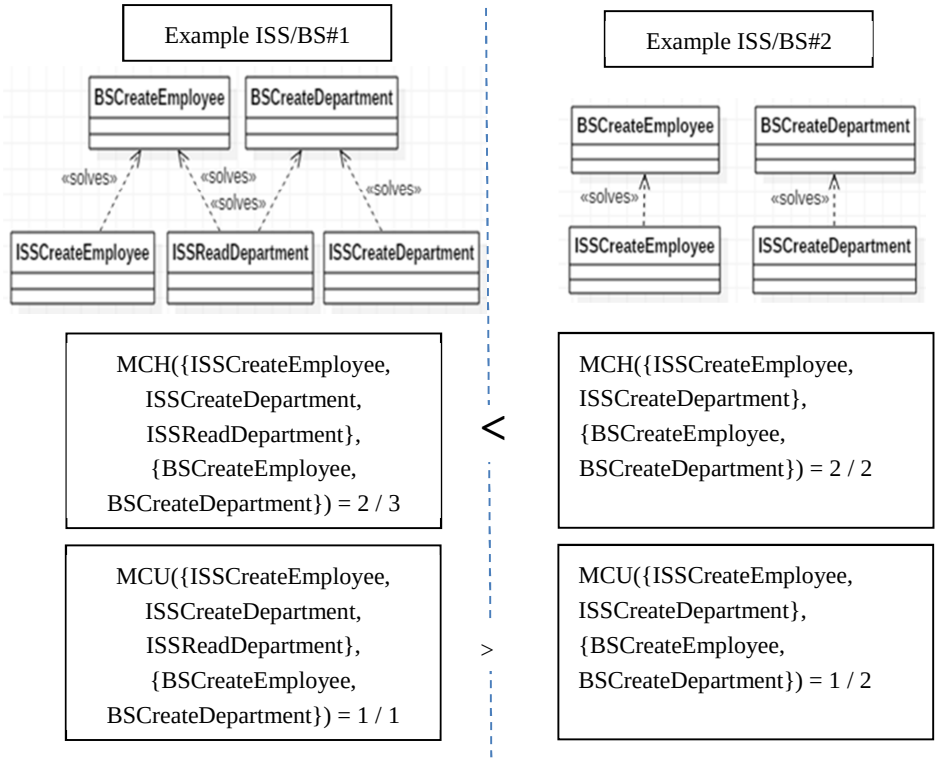

Fig. 4 – Computing coherence and coupling for IS services supporting business services.

Conclusion: Adding an information system service that partially resolves at least two uncoupled business services decreases the value of the coherence and increases the value of the coupling. That means that for a packaging system, the IS service sharing should not be considered when designing IS services. Duplication of elements, defining IS services, is thus possible if these elements support an identical task of a business service. However, as described in the following subsection, the IS service logical architecture will prohibit duplicating code when implementing these IS services. Obviously, the packaging system definition enforces this desirable behavior.

3.1.2.2. Coherence and coupling evaluations for LAC required by ISS

For the illustration of the logical application components coherence (see Eq. 1) and coupling (see Eq. 2) evaluations, two examples are proposed (see Fig. 5). LAC/ISS#1 solves two IS services with two LACs, one for one. In LAC/ISS#2 example, *LACDepartmentManagement* component solves both IS services.

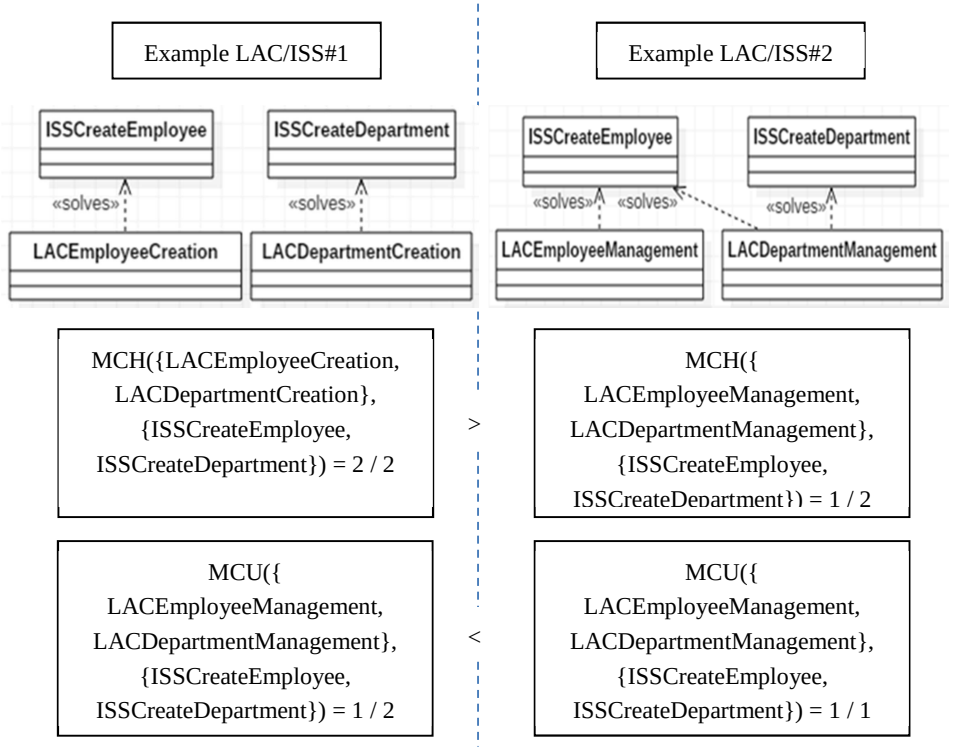

Fig. 5 – Computing coherence and coupling for LACs required by business services.

In LAC/ISS#1 example, the design of *LACEmployeeCreation* (reading of a department and creation of an employee) and *LACDepartmentManagement* (reading and creation of a department) involves a maximal coherence of the two LACs with the two IS services as each LAC solves only one specific IS service. In that case, it is important to note that the coherence MCH is maximized because each LAC implements the "department reading" specified in both IS services. The result of partitioning LAC sets obviously gives two complementary subsets involving a low MCU coupling measure.

About LAC/ISS#2, a lower coherence MCH of *LACEmployeeManagement* (creation of an employee) and *LACDepartmentManagement* (reading and creation of a department) is observed because *LACDepartmentManagement* solves both IS services. On the contrary, the coupling value MCU is maximal because only one LACs subset exists.

Conclusion: Adding a logical application component solving at least two IS services decreases the value of the coherence and increases the value of the coupling. That means

that for a packaging system, sharing of logical application components in order to solve IS services improves the packaging system properties.

3.1.2.3. Designing a system-of-services satisfying packaging system properties

Considering the previous examples and their MCH / MCU evaluations for both levels, the best design for the full system-of-services is given in Fig. 6 combining ISS/BS#2 and LAC/ISS#2, in order to comply with Definition 1 of a packaging system.

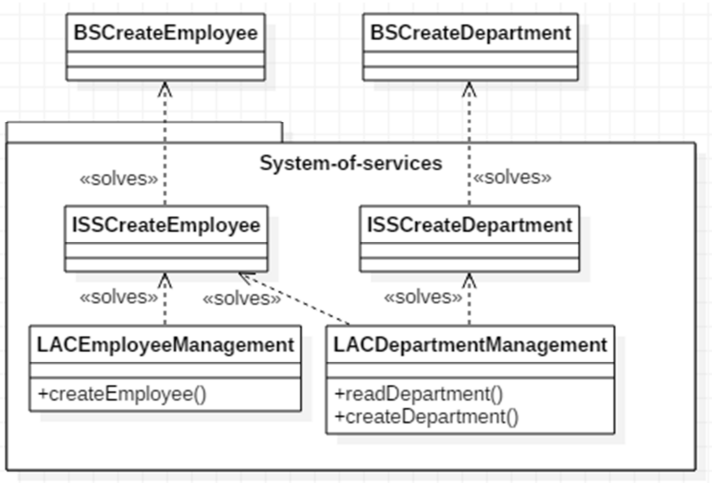

Fig. 6 – Illustration of system-of-services satisfying packaging system properties.

Thus, for a system-of-services, alternative designs for business services, information system services or logical application components can be deduced from:

- the increasing variation in the measure of coherence and the decreasing variation in the measure of coupling of information system services,
- the decreasing variation in the measure of coherence and the increasing variation in the measure of coupling of logical application components.

Such alternative designs are discussed about the use case in Section 5.1.4. An appropriate design solution shared by the different contributors (business expert, system-of-services logical architecture designer, IS service logical architecture designer) can then be extracted from these alternative designs.

Logical architecture is therefore central to our objective of generating IS services from business services. The difficulty in designing such architecture is the multiplicity of logical application components and the complexity of the dependencies between them. In order to reduce this complexity, we propose a pattern responding to the problem of designing the logical architecture of an IS.

3.2. *System-of-Services-Logical-Architecture-Design pattern*

The achievement of logical architecture design of a packaging system is enriched by models and concepts defining process patterns.[60] A process pattern is a pattern as defined which specifies activities for development.[61] These process patterns must solve a recurrent problem with activities description.[62] Activities description must contain activity chaining and enterprise responsible for each activity. Development recurrent problem studied in this section is the logical design of a system-of-services. The ***System-of-Services-Logical-Architecture-Design*** pattern is first detailed in Table 1 for the general characteristics of the pattern, in Table 2 for the process supporting the logical architecture design of system-of-services, and in Table 3 for the implementation.

Table 1 – System-of-Services-Logical-Architecture-Design pattern – General characteristics.

Category	***System-of-Services-Logical-Architecture-Design***
Classification	Scale: logical architecture model (LAC and LACD) description. Phase: logical architecture design. Purpose: logical architecture model design method for system-of-services having packaging system properties. Scope: logical architecture.
Intent	1) Design of the LACs defining a system-of-services and satisfying to a typing of LACs and 2) design of dependencies between LACs where orientation depends on the types of the source component and of the target component of each dependency.
Motivation	Scenario illustrating the problem is the development of a service, which belongs to a system-of-services having the properties of a packaging system. The development should be based on the logical architecture of the system-of-services in order to make easier the reuse of the services of the packaging system. The logical architecture of a system-of-services should minimize the coupling between services composing the system and supporting a business service (high coherence and low coupling).
Applicability	Logical architecture model of a system-of-services has beforehand to be aligned with business services that it supports.
Consequences	This pattern assists the logical architecture designer of a system-of-services and enables a use of the LAC model in any development of a service of the system-of-services.
Related Patterns	Analysis process patterns (before this pattern, for the business requirements analysis). Detailed design patterns (after this pattern, for the physical architecture design).

Process, in Table 2, supporting the logical architecture design of system-of-services is useful to a physical architecture designer of a service (transformation of a logical design into a physical design). The design of these activities follows the experimentation of the pattern in the field. The granularity of these activities is intended to implement the pattern directly through the precision of each of them. A logical data is defined as produced by a logical application component in the process.

Table 2 – System-of-Services-Logical-Architecture-Design pattern – Process.

Process	Following process activities and their chaining are represented in a UML activity diagram (see Fig. 7): • *Specification of LAC types from life duration criterion*: the system-of-services logical architecture designer defines first some types associated to LACs consistently with the life duration of these components. A typing is relevant for large scale system like Information System because the great number of LACs defining the system logical architecture. • *Design of logical data from business semantic*: in order to design the LACs, the logical architecture designer first designs a candidate logical data, with their attributes. They have to be aligned with the business services supported by the system of systems. The designer can be assisted in this activity by the business requirements analyst for the business understanding of the business services. • *Checking of logical data attributes life duration for each data*: the logical architecture designer checks that the attributes of the same candidate logical data satisfy the same life duration criterion associated to a type. • *Splitting of logical data*: if the attributes of a candidate logical data do not satisfy the same life duration criterion associated to a type, then the logical architecture designer must split the candidate logical data in order to design effective logical data having each some attributes conforming to one criterion. • *Design and typing of LAC defined each as managing one logical data*: if the attributes of each effective logical data satisfy the same life duration criterion, the logical architecture designer can design one LAC producing one effective logical data. • *Design of non-oriented LACDs from business semantic*: the logical architecture designer designs some dependencies that are non-oriented (i.e. not yet oriented) between the LACs. They have to be aligned with the business services supported by the system of systems. The designer can be assisted in this activity by the business requirements analyst for the business understanding of the business services. • *Orientation of LACDs from life duration rule*: for each non-oriented dependency, which associates two LACs having a different type, an orientation of the dependency is designed such a LAC having the type defined by the longest life cycle duration depends on the LAC

	having the type defined by the shortest life cycle duration. This rule is referred subsequently to as the ***Component-Dependency-BasedOn-Life-Duration-Rule***. • *Orientation of LACDs from business semantic*: for each non-oriented dependency, which associates two LACs having an identical type, an orientation of the dependency is designed such a component needing another component in the specification of business services depends on it. The designer can be assisted in this activity by the business requirements analyst for the business understanding of the business services. • *LAC cycle checking*: at last, the logical architecture designer checked that there is no cycle in the LAC model (cycle definition: there is a LAC1 LAC, which depends on a LAC2 LAC (different from LAC1), which depends on …, which depends on LAC1).
Participants	*Business requirements analyst*: analyst role is to assist the logical architect for the business understanding of the business services. *Logical architecture designer*: this architect has here to design a static logical architecture model of the system-of-services. The logical architecture designer is responsible for all the logical architecture activities specified in the process. *Physical architecture designer*: this architect must design the physical architecture of a service, part of the system-of-services, which implements a set of LACs and LACDs. These components and dependencies are collected by the physical architecture designer from the logical architecture of the system-of-services. Moreover, the physical architecture is deployed on technology architecture.

The process representation in Fig. 7 shows two roles. The triggering role is the *physical architecture designer* role, who is responsible for the physical implementation of the logical architecture of IS services. The contributing role is the *logical architecture designer* role, which has to design a static logical architecture model of the system-of-services.

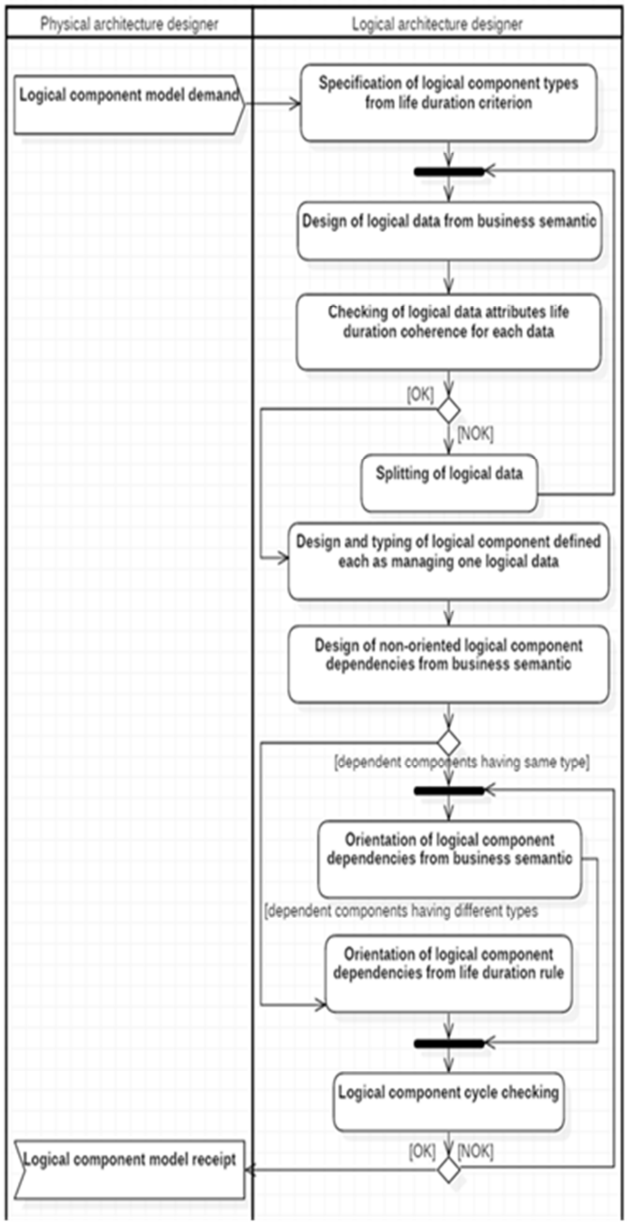

Fig. 7 - System-of-Services-Logical-Architecture-Design process.

The implementation of the pattern deals with the logical architecture model checking based on MDA approach. The concepts used by the transformation (LAC and logical application component dependency (LACD)) conform to the TOGAF meta-model's concept (Application architecture).[7] The model transformation is implemented with operational-QVT Language.[63] The implementation targets the two process activities:

Orientation of LACDs from life duration rule and *LAC cycle checking*. Indeed these two activities are the only ones that can be automated. The others require the intervention of the logical architecture designer, necessarily, and, optionally, of the business requirements analyst.

A pattern implementation targeting the data proposed 3 types: *Flow*, *Stock* and *Category*.[64] This pattern is adapted to LACs and experimented in several cases (teaching and research). Following these experiments and the application of the ***Component-Dependency-BasedOn-Life-Duration-Rule***, the *Flow* type becomes the *Activity* type, the *Stock* type is split into 2 types: *Person* and *Document*, and finally the type *Category* becomes the type *Reference*.

Table 3 – System-of-Services-Logical-Architecture-Design pattern description – Pattern implementation.

Implementation	For this implementation, the duration life is specified with the following typing of the LACs: • *Activity* that is associated with the management of dates or references of a flow modifying a status of a document, a status of a person or a reference, such as ordering a product or invoicing it is associated with a very short life cycle. • *Document* that targets documents such as commercial contracts associated with a product have a longer life cycle because the states of a document are changed by several activities, for example the order puts the contract in the created state and a service update puts the contract in the changed state. • *Person* which is associated to people such as the customer with, for example, a first order that puts the customer in the created state, while a second order with the same customer places him in the read state. In addition, the life cycle associated with the *Person* type is longer than that associated with the *Document* type, for example a customer may have several contracts, overlapping or not. • *Reference* which manages references such as those of a product in a catalogue, a type of commercial contract (after-sales service, etc.) or a customer market (company or general public) is associated with a longer life cycle than the previous types since it allows referencing the three previous types. The ***Component-Dependency-BasedOn-Life-Duration-Rule*** is adapted to this typing. Based on the assumptions about the life cycles associated with each type, the pattern used for the implementation contains six dependencies between the different types of LAC (see Fig. 8): • From *Activity* to *Document*, such as from a product order to a contract associated with a product. • From *Activity* to *Person*, such as from a product order to the customer who placed the order. • From *Activity* to *Reference*, such as from a product order to a service in the product catalogue. • From *Document* to *Person*, such as a contract associated with a

product to the customer holding the contract.
- From *Document* to *Reference*, such as a contract associated with a product to a type of contract defined by the seller.
- From *Person* to *Reference*, such as a customer to the market to which he belongs (general public or company for example).

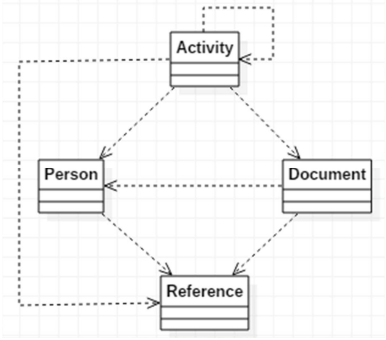

Fig. 8 - System-of-Services-Logical-Architecture-Design pattern for implementation.

The principle induced by this implemented pattern is that any *document* or *person* is created in the course of an *activity* and that the following *activities*, which are triggered by this *document* or *person*, depend on the *activity* of creation, and not on the *document* or *person*. For this reason, there is no dependency of logical application component, which is not typed *Activity*, on a logical application component, which is typed *Activity*, in the pattern. The implementation of the checking of the logical architecture of the system-of-services is in Appendix A.

Sample Execution

The execution samples run three cases of design of dependency between LACs.
The first case (see Fig. 9) does not satisfy the pattern about the orientation between components having different types.

Fig. 9 - Illustration of a logical architecture model not satisfying the *System-of-Services-Logical-Architecture-Design* pattern for implementation (see Fig. 8).

The execution result indicates a warning in relation with the pattern.

```
<terminated> lea2chk-example [Operational QVT Interpreter] In-process runner
Begin of the logical application component model checking

Orientation of logical component dependencies from life duration rule
WARNING: the dependency from LACDepartmentManagement to LACEmployeeManagement does not satisfy the pattern : person to reference

Logical component cycle checking

End of the logical application component model checking
```

The second one is an illustration of a loop (see Fig. 10), which concerns components having the same type.

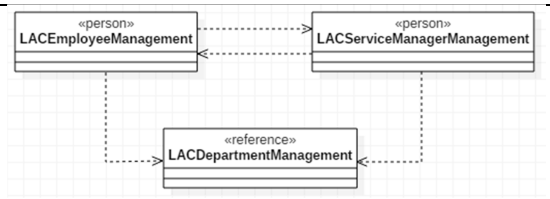

Fig. 10 - Illustration of a logical architecture model with a loop between *LACEmployeeManagement* and *LACServiceManagerManagement*.

The execution result indicates also warning meaning loops.

```
<terminated> lea2chk-example [Operational QVT Interpreter] In-process runner
Begin of the logical application component model checking

Orientation of logical component dependencies from life duration rule

Logical component cycle checking
WARNING: there is a logical loop from LACEmployeeManagement to itself
WARNING: there is a logical loop from LACServiceManagerManagement to itself

End of the logical application component model checking
```

The third one in Fig. 11 is a correct application of the pattern.

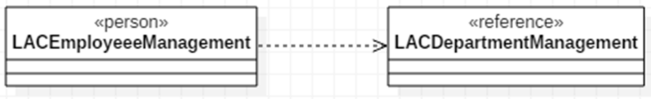

Fig. 11 – Illustration of a logical architecture model satisfying the *System-of-Services-Logical-Architecture-Design* pattern for implementation (see Fig. 8).

The execution result indicates no warning:

```
<terminated> lea2chk-example [Operational QVT Interpreter] In-process runner
Begin of the logical application component model checking

Orientation of logical component dependencies from life duration rule

Logical component cycle checking

End of the logical application component model checking
```

The design pattern of the logical architecture of a service system meets only functional expectations. The logical architecture addressing non-functional requirements is processed in the technical architecture model, where a technical component satisfies one or more non-functional requirements (mainly for response time). This part is not detailed here, but is implemented in the transformation of the logical model into a physical model of the IS services in a classical way in the MDA approach, i.e. with a PDM representing this technical architecture (see 4.2).

3.3. *Information system service definition*

The definition of an IS service (the same concept as in the TOGAF meta-model) is based here on the properties of a logical dependency tree (LDT) made of LACs.

Definition. A set of LACs LDT = {LACi such $1 \leq i \leq n$} is a ***logical dependency tree*** if it conforms to an acyclic directed graph having one root: $\exists$! LACroot $\in$ LDT such $\forall$ LACi

∈ LDT with LACi ≠ LACroot ⇒ *path*(LACi, LACroot) = *false* and *path*(LACroot, LACi) = *true*.

The logical architecture in Fig. 11, which satisfies the ***System-of-Services-Logical-Architecture-Design*** pattern, is a ***logical dependency tree*** where LACroot = LACEmployeeManagement with path(LACEmployeeManagement, LACDepartmentManagement) = *true* and path(LACDepartmentManagement, LACEmployeeManagement) = *false*.

In order to express the coupling property of LACs composing an IS service, IS service definition is based on a ***logical dependency tree***.

Definition. The logical architecture of an IS service, which has to support all or part of a business service, is defined by LACs and LACDs that form a ***logical dependency tree***.

An illustration is the definition of the IS service supporting the *BSCreateEmployee* business service. The specification of *BSCreateEmployee* results in two business tasks:

(i) Read a department
(ii) Create an employee

The ***logical dependency tree*** in Fig. 11 allows to define *ISSCreateEmployee* as an IS service. The instantiation of LACs' tree is underlined by a UML sequence diagram representing an instance of the *ISSCreateEmployee* IS service (see Fig. 12).

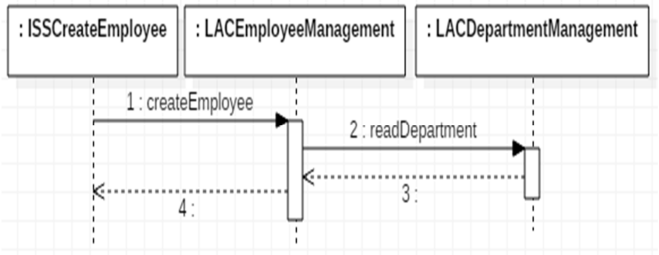

Fig. 12 - IS service supporting *BSCreateEmployee* business service.

The ***logical dependency tree*** is completely instantiated because the two LACs are instantiated, and because the LACD from *LACEmployeeManagement* on *LACDepartmentManagement* is instantiated by the *readDepartment* request operation instantiation. The *createEmployee* logical operation of *LACEmployeeManagement* and the *readDepartment* logical operation of *LACDepartmentManagement* are instantiated when *ISSCreateEmployee* is instantiated.

The ***logical dependency tree*** in this example fully supports the *BSCreateEmployee* business service. However, the LACs supporting a business service could be associated to more than one ***logical dependency tree***, and thus define more than one IS services (one IS service per one ***logical dependency tree***). Moreover, each LAC can support more than one business service. In order to address this complexity, we define the LAC model of the system-of-services as a contextual model of the transformation of business services into IS services.

Each ***logical dependency tree*** design should conform to the logical view of the packaging system. The generation of a IS service from a business service must be thus

based on the logical architecture of the packaging system, which is the IS containing it and which conforms to the ***System-of-Services-Logical-Architecture-Design*** pattern.

4. IS Service Generation Algorithm for IS Service Development

Knowing the logical architecture of a service system with high consistency properties induced by the supported business services, the problem to be solved is to generate IS services, from business services, with a low coupling between them, and so that each service is composed of LACs with a high coupling and a low coherence between them.

The definition of an IS service must then highlight the coupling property of the LACs that describe it. One solution consists in constraining an IS service by the existence of dependencies between its LACs. In order to define the IS services that comply with this solution, an IS services generation algorithm, based on a set of business services, is proposed in this section. The model-driven approach to using a logical model of the service system resulting from the pattern (see Section 3.2) and reusing existing IS services completes this section.

4.1. *Contextual model transformations from business services to IS services*

The complexity highlighted above can be considered as a problem of variability of "data" related to an IS service development process. Indeed, for each business service, the IS service architecture designer must 1) align this service with the service system logic model in order to automatically generate the IS services supporting the business service, 2) align the logical architecture of the IS services supporting the business service with one or more existing IS services of the system-of-services (external to the transformation) in order to reuse them (conditioned by technological consistency in the subsequent design of the physical architecture). The IS service architecture designer must therefore align 1) a model of LACs of the service system (up to several hundred components, each supporting one or more business services) with the tasks of a business service, and then 2) existing IS services (up to several thousand) with the logical architecture of a business service. These numerous "data" form a double context that induces 1) an integration with enrichment by a logical model of the service system, and 2) an integration with substitution by existing IS services.

The model-driven integration (useful for these many "data") of this double integration is proposed with a MDA-compliant development process. The double integration is based on the integration of a context into a MDA model transformation.[25] In this approach, the authors integrate a contextual model (TCM: Transformation Context Model) into the PIM before ST (Substitution Transformation) or ET (Enhancement Transformation) applications resulting in a PSM under the technical constraints of a PDM. The PICM (Platform Independent Contextual Model) results from the integration of the TCM into a PIM.

This integration must be extended to the full MDA approach. The business services model is indeed a CIM (Computation Independent Model) in the MDA approach. This means an adaptation to make of the contextual transformation (substitution or

enhancement)[25] for the transformation of a CIM into a PIM. The enhancement of CIM with the logical architecture of the service system (TCMe: TCM for enhancement) leads to the logical architecture model of IT services supporting business services (PIM). Before obtaining the PIM, we propose an enriching CTe contextual transformation of the CIM.

4.2. *Enhancement contextual transformation of business services into IS services logical model (CTe)*

The CIM is a meta-model of a business service. The CIM concepts target the description of a business task composing a business service. These concepts takes into account some constraints making easier the generation of IS services code. CIM's concepts definition and illustration in relation to the example in Section 4.1, are as follows:

- "Business Service" (the same concept as in the TOGAF meta-model) describes a business service (*BSCreateEmployee*).
- "Business Task" specifies the order number of a task composing the business service (*1* for the *Read a department* business task of the *BSCreateEmployee* business service and *2* for the *Create an employee* task).
- "Verb" indicates the verb defining the business task (*read* for the *Read a department* business task and *create* for the *Create an employee* task). The verb value is limited to *create*, *read*, *update*, and *delete* in order to make easier the transformation resulting in code (with reference to CRUD designating the four basic operations for data persistence).
- "Data Entity" (the same concept as in the TOGAF meta-model) represents a data associated to a business task. The association means the production of a data by a business task (*Department* for the *Read a department* task and *Employee* for the *Create an employee* task).
- "Attribute" refers to an attribute of the data entity, which is associated to the business task. This attribute is directly related to the task (*name* attribute of the *Department* data entity for the *Read a department* business task and *name*, and *social security number* attributes of the *Employee* data entity for the *Create an employee* task. The attribute is essential for a task associated with a test. In this case, only the attribute of the data entity used for the test is associated with the task. For example, when selecting an employee, only the *social security number* is associated with the reading task.
- "Loop" indicates if the business task is iterated or not. The loop value is *true* or *false*.
- "Condition" describes the condition that is associated to a business task. A condition targets an attribute characterizing a task, which specifies a test (for example, *<>null* condition for *name* attribute of the *Department* data entity for the *Read a department* business task).

TCMe is the contextual model in relation to the integration with enhancement. This enhancement by a logical architecture model of the system-of-services needs the following concepts of the TCMe meta-model:

- "Logical Application Component" (*LACEmployeeManagement*) and "Logical Application Component Dependency" (*LACEmployeeManagement* on

LACDepartmentManagement) as used in the ***System-of-Services-Logical-Architecture-Design*** pattern.

The CICM (Computation Independent Contextual Model) meta-model shows a mapping relationship between a business service and a system-of-services logical model. This mapping is achieved with the concept "Contextualized Data Entity Attribute" that links the attribute of a data entity (concepts "Data Entity" and "Attribute") with a LAC (concept "Logical Application Component") (see Fig. 13). The attribute selection is implicitly constrained by its membership of the entity data. This mapping between units (represented by "→") specifying a business service and units characterizing a system-of-services logical model is extended to a mapping between relationships (represented by "→"). A sequence of business tasks (see the concept "Business Task" composed of the order number of the task) involves a relationship between the associated data entity attributes. The rule ***CTe-RR*** defining a mapping of this relationship between the associated data entity attributes with a LACD is as follows.

CTe-RR. Let the contextual CTe-Transformation from CIM x TCMe to CICM, *DE1 – A1* and *DE2 – A2*, two data entity attributes associated to two business tasks in CIM, resp. *BT1* and *BT2* such *BT1* comes before *BT2*, if ∃ *LAC1* and *LAC2* in TCMe such:

CTe(*DE1 – A1*, TCMe) = *DE1 – A1* → *LAC1*

CTe(*DE2 – A2*, TCMe) = *DE2 – A2* → *LAC2*

and if ∃ a dependency {*LAC2* on *LAC1*} in TCMe, then

CTe({*DE1 – A1* before *DE2 – A2*}, TCMe) =

{*DE1 – A1* before *DE2 – A2*}→ {*LAC2* on *LAC1*}

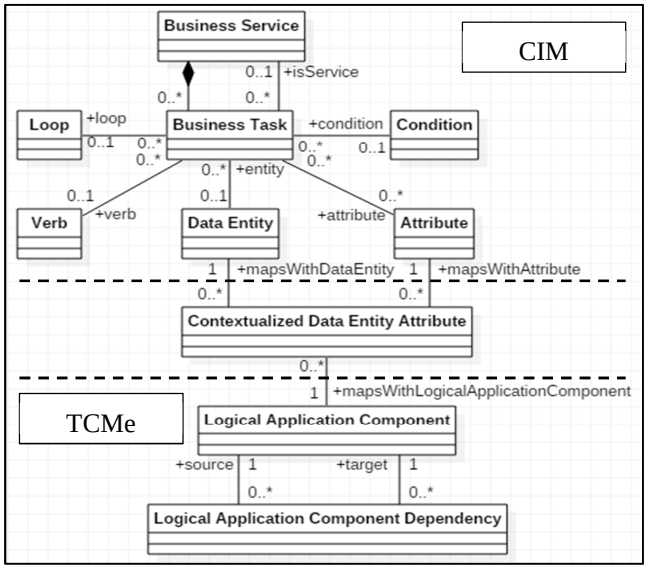

Fig. 13 – CICM meta-model including the mapping between CIM concepts and TCMe concepts.

The contextual CTe-Transformation (from CIM x TCMe to CICM) is illustrated such*:*

CTe(Department - name, {LACEmployeeManagement, LACDepartmentManagement})
= Department - name → LACDepartmentManagement
CTe(Employee - name, {LACEmployeeManagement, LACDepartmentManagement}) =
Employee - name → LACEmployeeManagement
CTe(Employee - social security number, {LACEmployeeManagement,
LACDepartmentManagement}) =
Employee - social security number → LACEmployeeManagement

CTe is applied to the CIM task sequence: *1. Read a department* before *2. Create an employee*. This task sequence means a sequence: the *Department* data entity and its attribute *name* before the *Employee* data entity and its attributes *name* and *social security number*. From the ***CTe-RR*** rule, the relationship *Department - name* before *Employee – name & social security number* maps with the LACD: *LACEmployeeManagement* on *LACDepartmentManagement*:

CTe({Department - name before Employee – name & social security number},
{LACEmployeeManagement on LACDepartmentManagement}) =
{Employee – name & social security number on Department – name} →
{LACEmployeeManagement on LACDepartmentManagement}

On the one hand, this mapping must be done by the IS service architect. However, this architecture designer needs business knowledge useful for understanding the attributes of the data entities produced by the business tasks. The integration with enhancement is thus an activity of experts (core business of the company and logical architecture of the system-of-services) that target a domain model enhancement with a mapping between the vocabulary that is familiar to the business core practitioners of the domain and the vocabulary that is used by the domain's IS practitioners. This activity cannot thus be automated. This is why CICM retains the computational independence properties of CIM.

On the other hand, the transformation of CICM resulting in the logical architecture of the IS services (ET transformation) is automatized with the implementation of the following algorithm of generation of IS services from a set of business services.

4.3. *Algorithm of generation of information system services from business services (ET)*

We propose in this section an algorithm of generation of IS services from a set of business services. The only rule that the business expert must satisfy here is a temporal scheduling of the tasks composing the business service, without conditional connection (*if then else* instruction, for example). Each conditional connection involves a business service (associated to the *if then* part of the instruction and another business service associated to the *else* part, in the example).

The generation of IS services is based on the definition of a ***logical dependency tree*** (see 4.1). Each business task is associated to an attribute of a data entity. Consistently with the definition of a context associated to an attribute of a data entity in CICM, each business task can be associated with one or more LACs. In relation to a set of business

tasks composing a business service, a set of LACs is thus instantiated for each business service. In a simplified way, the splitting into IS services, implemented in ET, and supporting this business service is based on the greatest logical dependency trees composing this set of LACs. The reuse of the IS services resulting from the transformation is also addressed by ET, and the order of the business tasks describing a business service increase the complexity of this splitting.

The precondition of the ***Generation of Information System Services from Business Services*** algorithm is the availability of CICM. The post condition if the set of IS services that are instantiated for a set of business services specified with its enhancement context in CICM.

The input data of the Generation of Information System Services from Business Services algorithm are:

- *BSS*, a set of business services, each composed of business tasks (CIM).
- *SoSLAC*, a set of *LACs* of the system-of-services, and *SoSLACD*, a set of *LACDs* of the system-of-services that both result from the ***System-of-Services-Logical-Architecture-Design*** pattern application (TCMe).

The output data of the algorithm are:

- The *ISSIS* that is a set of instances of IS services supporting a business service. The reuse of generated IS service(s) is carried out by the algorithm (PIM).
- The *LOIS*, which is a set of logical operation instances such each logical operation composes a LAC of *SoSLAC* and such the set *LOIS* is useful to an instance of an IS service of *ISSIS* (PIM).

The algorithm and the needed functions are described in Fig. 14. The algorithm is written with pseudo-code language close to operational-QVT language.

Functions
// Returns ISSIS including an instance of a generated IS service, which can be reused,
// instantiating a LOIS set of logical operation instances.
update_information_system_instance (LOIS, ISSIS);
// Returns the attributes, associated to BT task that are 1) mapped with LACs of SoSLAC,
// 2) defined by data entity attributes' context (CICM in Fig. 13), such these LACs and
// the LACs encapsulating the instantiated logical operation of LOIS, compose ***a logical***
// ***dependency tree*** (see 4.1), knowing the dependencies of SoSLACD.
test_mapping (LOIS, BT);
// Returns LOIS including the instances of the logical operations defined by 1) the context
// (LACs) of the data entity attributes associated to BT (CICM in Fig. 13) and 2) the verb
// characterizing BT
update_logical_operation_instance (BT.attributes, LOIS);
EndFunctions

Algorithm
Generation of Information System Services from Business Services
BSS : Set(Business Service) := {BS defining a business service};
// *SoSLAC* and *SoSLACD*, resulting from the
// ***System-of-Services-Logical-Architecture-Design*** pattern

```
SoSLAC : Set(LAC) := {LAC ∈ System-of-Services};
SoSLACD : Set(LACD) := {LACD ∈ System-of-Services};
// Generation of Information System Service(s) from the Business Service(s) set
BS : Business Service := BSS->first();
// Iteration for each BS business service of BSS
while (BS<>null)
{
  // Initialization of ISSIS and OIS to empty set for each BS
  ISSIS : Set(Information System Service Instantiation) := Set{};
  LOIS : Set(Logical Operation Instantiation) := Set{};
  // Selection of the last (temporal) BT task of BS
  BT : Business Task := BS.businessTasks->last();
  // Iteration (last to first one) for each BT task of BS
  while (BT<>null)
  {
    if (test_mapping (LOIS, BT) = BT.attributes)
    then
    // Update of the logical operation instance set without yet the generation of an
    // information system service instance
      LOIS := update_logical_operation_instance (BT.attributes, LOIS)
    else
      // The attributes conforming to a logical dependency tree do not recover all the
      // attributes of BT
      if (test_mapping <> {})
      then
      {
        // Completion of LOIS with logical operation instances associated to attributes
        // conforming to the test of mapping
        LOIS := update_logical_operation_instance (LOIS, test_mapping);
        // Update of ISSIS based on updated LOIS including the reuse or the generation of
        // one IS service
        ISSIS := update_information_system_instance (LOIS, ISSIS);
        // Initialization of LOIS with the attributes of BT that do not satisfy the test of
        // mapping
        LOIS := update_logical_operation_instance (BT.Attributes – test_mapping, {});
      }
      else
        // Update of ISSIS based on updated LOIS including the reuse or the generation of
        // one IS service
        ISSIS := update_information_system_instance (LOIS, ISSIS);
      endif
    endif;
    BT := BT.previous;
  }
  endwhile;
  // Update of ISSIS based on updated LOIS including the reuse or the generation of one
  // IS service
  ISSIS := update_information_system_instance (LOIS, ISSIS);
```

```
  BS := BS.next;
}
endwhile;
EndAlgorithm
```

Fig. 14 – Generation of Information System Services from Business Services algorithm and associated functions.

The illustration of the algorithm running is based on the example of PICMe (see 4.2):

- BSS = {*BSCreateEmployee*}
- SoSLAC = {*LACEmployeeManagement, LACDepartmentManagement*}
- SoSLACD = {*LACEmployeeManagement on LACDepartmentManagement*}
- LOIS = {}
- ISSIS = {}

The first iteration targets the business task BT = *2. Create an employee* with

CTe(Employee - name, {LACEmployeeManagement, LACDepartmentManagement}) = Employee - name → LACEmployeeManageement

CTe(Employee - social security number, {LACEmployeeManagement, LACDepartmentManagement}) = Employee - social security number → LACEmployeeManageement

⇒ test_mapping ({}, {*Employee - name*, *Employee - social security number*} = {*2. Create an employee*}.attributes (i.e. the attributes associated to the business task)

⇒ LOIS = {*createEmployee* instance} and ISSIS = {}.

The second iteration targets the business task BT = *1. Read a department* with

CTe(Department - name, {LACEmployeeManagement, LACDepartmentManagement}) = Department - name → LACDepartmentManagement

⇒ test_mapping ({createEmployee instance}, {Department - name} = {1. Read a department }.attributes

because LACEmployeeDepartment, context of Employee attributes, and LACDepartmentManagement, context of Department attribute, form a logical dependency tree (see 4.1).

⇒ LOIS = {*createEmployee* instance, *readDepartment* instance}

⇒ ISSIS = {*ISSCreateEmployee* instance} designed as illustrated by the sequence diagram in Fig. 11.

The low coupling between IS services is directly deduced from the *test_mapping* function of the algorithm which split a BS service into IS services because a lack of dependency of the system-of-services in order to form a ***logical dependency tree***, and therefore a lack of coupling. However, this property induces a coupling between the instantiated LACs designed for an IS service instantiation. Coherence as defined in Section 3.1 is low due to the partial coverage of a business process (triggered by an external expectation) by the LACs of an IS service.

The ***Generation of Information System Services from Business Services*** algorithm allows to generate the IS services, which are designed with the logical architecture model of the system-of-services. The implementation of the algorithm automates this design.

The automation of IS services development also requires the automation of the transformations of 1) the logical architecture model into a physical architecture model, and 2) of this physical architecture model into the code.

4.4. *Substitution contextual transformation of IS services logical model into IS services physical model (CTs)*

In order to deal with the ST logical architecture model transformation into a physical architecture model, a contextual transformation allows the reuse of IS services existing before in the system-of-services. This reuse means a contextual transformation by substitution. The substitution relates to an excerpt of the logical operations instantiated by a new IS service, which map with the logical operations defining the logical architecture of an existing IS service.

TCMs (TCM for substitution) meta-model, which is the contextual model for this integration with substitution, contains only one concept:

- "Information System Service" as defined in Section 4.1. An IS service describes only an existing service (external to the transformation). The instantiation of this concept means that the service can be reused.

The PICM meta-model,[25] contains a concept "Contextualized Logical Operations" that enables a mapping between a set of logical operations and an existing IS service that instantiates them (see Fig. 15).

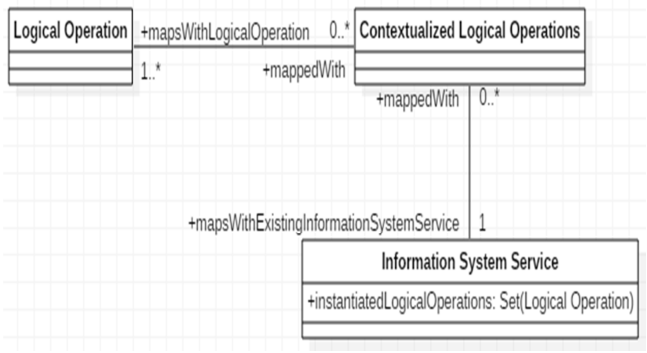

Fig. 15 – PICM meta-model including the mapping between PICM (Logical Operation) and TCMs (Information System Service).

The contextual CTs-Transformation is a function from PIM x TCMs to PICM. In order to illustrate it, The PIM's illustration is completed by one existing IS service called *ISSReadDepartment* that supports the *readDepartment* logical operation (notice that more than one IS service could support the same set of logical operations, with a specificity of each one in relation to its execution environment such those associated to Java or C++, for example):

$$\text{CTs}(\textit{readDepartment}, \{\textit{ISSReadDepartment}\}) = \textit{readDepartment} \rightarrow \textit{ISSReadDepartment}$$

Justified by the supported logical operation of *ISSReadDepartment*, the *readDepartment* logical operation of the *ISSCreateEmployee* business service can be associated to *ISSReadDepartment* element (represented by "→") in PICM.

By assumption, there is no existing IS service supporting a *createEmployee* logical operation:

$$CTs(createEmployee, \{ISSReadDepartment\}) = \{\emptyset\}$$

CTs cannot therefore be applied to the PIM relationship (logical operations dependency, for example) from *createEmployee* on *readDepartment*.

This mapping should be achieved by the architecture designer of the IS services. However, the attribute "instantiatedLogicalOperations", which is a set of logical operations, of the "Information System Service" TCMs' concept enables an automatization of the reuse of an IS service. This automatization is the result of the implementation of the ST transformation, which is not detailed in this section, because it is much more common in the MDA approaches implemented today. The MDA approach extended with the consideration of an enhancement context for the business model and a substitution context for the logical architecture enables to log each step of the development. The automatic modeling of the deliverables (architecture and code) highlights the possibilities offered by the chaining of model transformations. This transformation chaining is applied in the following section to two uses cases about 1) management IS services and 2) real time IS services.

5. Use Case of IS Services Development from Business Services

The objective of the uses case is to check the properties of coherence and coupling defining a system-of-services as a packaging system. This checking is based below on the evaluation (see 3.1) of 1) the coherence by the ratio between the number of IS services (resp. LACs) that exhaustively support one business service (resp. IS service) and the total number of IS services (resp. LACs), and of 2) the coupling by the number equal to 1 divided by the minimum number of subset of IS services (resp. LACs) such that two IS services (resp. LACs) belonging to two different subsets cannot solve the same business service (resp. IS service). The use case illustrating a management IS is based on practical works on the alignment of an IS architecture with the business processes description for 2nd year of Masters, at university and in a postgraduate engineering school.

5.1. *Use case of development of management information system services*

The use case is practical works for the Information System Management Master of the Business Administration Institute (IAE) (Western Brittany University (UBO)). The system-of-services is a management IS dedicated to commercial services. These services support commercial relationship management (CRM) and invoicing management. The business process to be developed is a purchasing process in a large household appliance store.

5.1.1. *Commercial IS design from* ***System-of-Services-Logical-Architecture-Design*** *pattern application*

The commercial IS has to support the business activities of a large household appliance store. The logical architecture of this commercial IS (see Fig. 16) satisfies the ***System-of-Services-Logical-Architecture-Design*** pattern (see 3.2). The LACs stereotyped "activity" by the logical architecture designer of the commercial IS (role played by the teacher) manage date or reference of an order (*LACOrderManagement*), a payment (*LACPaymentManagement*), a (de)stocking (*LACStockingManagement*), and a *delivery* (LACDeliveryManagement). These activities depend on a customer managed by a LAC stereotyped "person" (*LACCustomerManagement*), and by a product managed by a LAC stereotyped "reference", which is relevant for a list of available products (*LACProductManagement*), and by payment means also managed by a LAC stereotyped "reference", which is appropriate to a list of accepted payment means by the store (*LACPaymentMeansManagement*). Dependencies that support the commercial business of the store satisfy the pattern in relation to the stereotypes of the LACs.

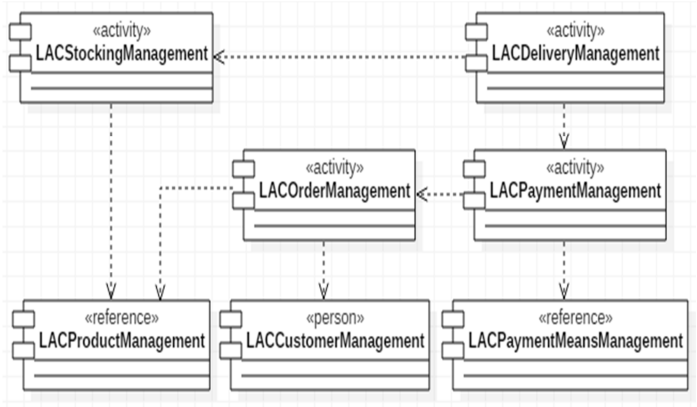

Fig. 16 – UML 2.2 commercial system-of-services logical model (TCMe).

This logical model of the commercial system-of-services drives the generation and design of IS services supporting business services. These services are designed by students playing business expert role.

5.1.2. *Commercial IS services generation from* ***Generation of Information System Services from Business Services*** *algorithm*

The business process represents the answer of the large household appliance store, split into departments (*sales*, *billing*, *supply*, and *delivery* departments), to a customer request about a list of products. All the activities, under the responsibility of a department, are inside the store. This means that the customer enters the store to place an order (*Create order* activity), pays (*Create bill and payment* activity) and exits with the purchased

product (*Create product delivery* activity), which of course has been destocked from the warehouse (*Destock one product activity*).

Business services are specified from the previous business process description. The specification rule of a business service without conditional connection described in Section 4.4 is applied to the first activity, *Create order*, where a conditional connection is associated to the preexistence of a customer of the large household appliance store. Two business services are thus deduced from this activity: *BSCreateOrderNewCustomer*, when the customer is new, and *BSCreateOrderExistingCustomer*, when the customer already exists for the store. The three other activities led to the three following business services:

- *BSCreateBillAndPayment* deduced from *Create bill and payment* activity.
- *BSDestockOneProduct* deduced from *Destock one product* activity.
- *BSCreateProductDelivery* deduced from *Destock one product* and *Create product delivery* activities.

Results of the transformations running, of the ***Generation of Information System Services from Business Services*** algorithm (see 4.4), are reported in Fig. 17.

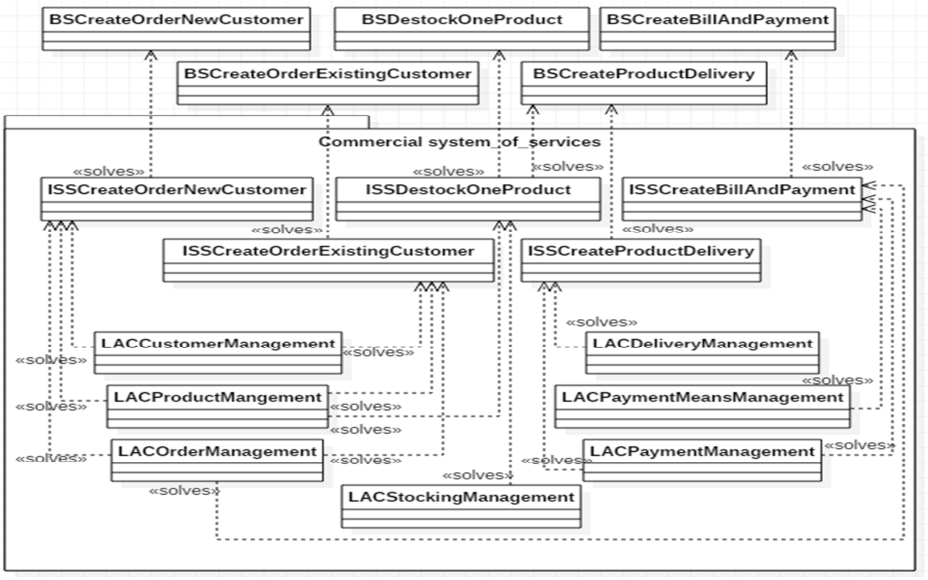

Fig. 17 – Illustration of a commercial system-of-services supporting business services dedicated to ordering, billing and delivery.

From the logical architecture of the IS services supporting the business process representing the store activities triggered by a customer's request of products, the transformations of models resulting in IS services code and database's generation SQL script can be applied. Notice that the design of the orchestration of the IS services supporting the business services sequence composing the process is not processed by the algorithm.

5.1.3. *Business service's loop transformation illustration*

To illustrate the commercial IS services automatized development, we propose to focus on the complete model transformations chaining of a loop specified in a business service. Fig. 18 represents the textual specification of *BSCreateOrderExistingCustomer*. A loop targets the second task, which is the reading of an ordered product. Notice that the business service specification must be checked beforehand by a business expert.

```
<Business Service><name>BSCreateOrderExistingCustomer</name>
<Business Task><order number>1</order number>
<Verb><crud>read</crud></Verb>
<Data Entity><name>Customer</name></Data Entity>
<Attribute><name>name</name><type>String</type></Attribute>
<Attribute><name>address</name><type>String</type></Attribute>
<Condition><guard><>null</guard></Condition>
</Business Task>
<Business Task><order number>2</order number>
<Verb><crud>read</crud></Verb>
<Data Entity><name>Product</name></Data Entity>
<Attribute><name>name</name><type>String</type></Attribute>
<Attribute><name>cost</name><type>String</type></Attribute>
<Loop><list>true</list></Loop>
</Business Task>
<Business Task><order number>3</order number>
<Verb><crud>create</crud></Verb>
<Data Entity><name>Order</name></Data Entity>
<Attribute><name>date</name><type>String</type></Attribute>
<Attribute><name>reference</name><type>Integer</type>
</Attribute>
</Business Task>
</Business Service>
```

Fig. 18 – *BSCreateOrderExistingCustomer* business service textual specification including a loop (enclosed).

The generation and the UML2 logical model design of the IS services needs an alignment of the CIM entity attributes participating to *BSCreateOrderExistingCustomer* with the TCMe LACs. Six contextualized entity attributes are designed in the CICM, by the students, in relation to *BSCreateOrderExistingCustomer*:

CTe(*Customer - name*, LAC_SO) = *Customer - name* → *LACCustomerManagement*

CTe(*Customer - address*, LAC_SO) = *Customer - address* → *LACCustomerManagement*

CTe(*Product - name*, LAC_SO) = *Product - name* → *LACProductManagement*

CTe(*Product - cost*, LAC_SO) = *Product - cost* → *LACProductManagement*

CTe(*Order - date*, LAC_SO) = *Order - date* → *LACOrderManagement*

CTe(*Order - reference*, LAC_SO) = *Order - reference* → *LACOrderManagement*

Fig. 19 illustrates the automatized transformation of the CICM into the PIM. The dynamic logical architecture of *ISSCreateOrderExistingCustomer:createOrder_date*, which is represented by a UML2 sequence diagram, highlights the dependencies from *LACOrderManagement* on both *LACCustomerManagement* and *LACProductManagement* designed in the commercial IS logical architecture (see Fig. 16). The UML2 combined fragment (boucle = loop in French) results from the transformation of the loop specified in *BSCreateOrderExistingCustomer* (see Fig. 18).

The transformation of the PIM into the PSM is contextualized by the existing IS services of the commercial IS. Only one IS service is available in the TCMs: *ISSReadAProduct:readProduct_name*, which implements the *readProduct_name* logical operation and is deployed on the same technical infrastructure than the *ISSCreateOrderExistingCustomer:createOrder_date* IS service. *ISSReadAProduct:readProduct_name* can therefore be immediately reused if one considers the dynamic logical model of *ISSCreateOrderExistingCustomer:createOrder_date*. The reuse is explicit in the dynamic physical model of this IS service (see Appendix B).

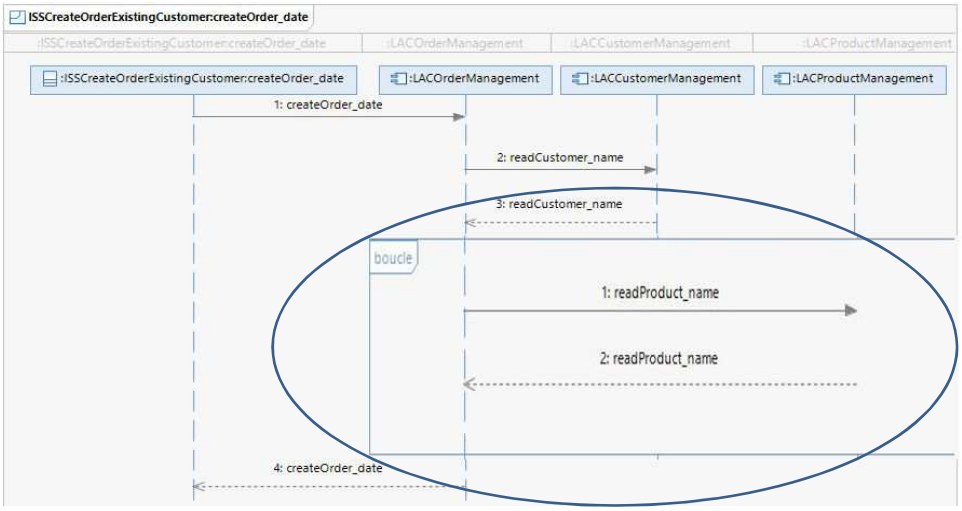

Fig. 19 – *ISSCreateOrderExistingCustomer:createOrder_date* IS service dynamic logical model transformed into UML2 sequence diagram including a loop combined fragment of the *readProduct_name* logical operation (enclosed).

One question is then whether the packaging system properties of the commercial system of services can be improved.

5.1.4. *Commercial packaging system-of-services*

An assessment of the coherence and the coupling, of the commercial system–of-services (see Fig. 17), is first made in order to propose an eventual improvement of the packaging system properties.

About the generated IS services, *ISSDestockOneProduct* solving two different business services (*BSDestockOneProduct* and *BSCreateProductDelivery*) is a cause of decreasing coherence.

Applying Eq. (1) with

BS_EX = {*BSCreateOrderNewCustomer*, *BSCreateOrderExistingCustomer*,
BSCreateBillAndPayment, *BSDestockOneProduct*,
BSCreateProductDelivery},

and

ISS_SO = {*ISSCreateOrderNewCustomer*, *ISSCreateOrderExistingCustomer*,
ISSCreateBillAndPayment, *ISSDestockOneProduct*,
ISSCreateProductDelivery}:

MCH(ISS_SO, BS_EX) = 4 / 5 = 0.80

The orchestration of *ISSDestockOneProduct* and *ISSCreateDelivery*, which solves the *BSCreateProductDelivery* business service, increases the coupling of ISS_SO conditioned by BS_EX. There are indeed four complementary subsets, including the one containing *ISSDestockOneProduct* and *ISSCreateDelivery*, and the ISS_SO's subsets containing only one IS service among the remaining three. According to Eq. (2):

MCU(ISS_SO, BS_EX) = 1 / 4 = 0.25

Regarding the LACs, *LACStockingManagement* solves only the *ISSDestockOneProduct* IS service, *LACPaymentMeans* solves only *ISSCreateBillAndPayment*, and *LACDeliveryManagement* solves only *ISSCreateProductDelivery*. These three LACs satisfy the coherence property.

Using Eq. (1), with:

ISS_EX = {ISSCreateOrderNewCustomer, ISSCreateOrderExistingCustomer,
ISSCreateBillAndPayment, ISSDestockOneProduct,
ISSCreateProductDelivery},

and

LAC_SO = {LACOrderManagement, LACCustomerManagement,
LACProductManagement, LACPaymentManagement,
LACPaymentMeansManagement, LACStockingManagement,
LACDeliveryManagement}:

MCH(LAC_SO, ISS_EX) = 3 / 7 = 0.43

There is none LAC that is coupled with none of the other LACs. According to Eq. (2):

MCU(LAC_SO, ISS_EX) = 1 / 1 = 1.00

These four measures highlight a non-maximum coherence of ISS_SO (<1.00) and a non-minimum coupling (> 1/5). About LAC_SO, the coherence is much higher than the minimum expected (0.00) when the coupling is optimal (1.00).

The coherence and coupling properties of this commercial system-of-services can only be improved from a business perspective. Concerning the large household appliance store, the delivery is only made inside the store. In this case, the business expert could consider only one business service, and thus one role, to accomplish the *Destock one product* activity and the *Create product delivery* activity. Fig. 20 shows the result of the generation of the IS services with a new *BSDestockOneProductAnd CreateProductDelivery* business service. The running the ***Generation of Information System Services from Business Services*** algorithm results in a new *ISSDestockOneProductAndCreateProductDelivery* IS service because there is a dependency from *LACDeliveryManagement* on *LACStockingManagement* (see Fig. 16). Without this dependency, the algorithm should indeed generate two different IS services

for the destocking on the one hand, and for the delivery on the other hand. The measures of coherence and coupling properties become such:

MCH(ISS_SO, BS_EX) = 4 / 4 = 1.00
MCU(ISS_SO, BS_EX) = 1 / 4 = 0.25
MCH(LAC_SO, ISS_EX) = 3 / 7 = 0.43
MCU(LAC_SO, ISS_EX) = 1 / 1 = 1.00

Compared to the first designed commercial system-of-services, the consideration of a delivery inside the store by the business expert leads to:

- A better coherence of ISS_SO (1.00 > 0.80), which is maximum;
- A better coupling of ISS_SO, which is minimum here;
- A constant coherence and a constant coupling of ISS_LA.

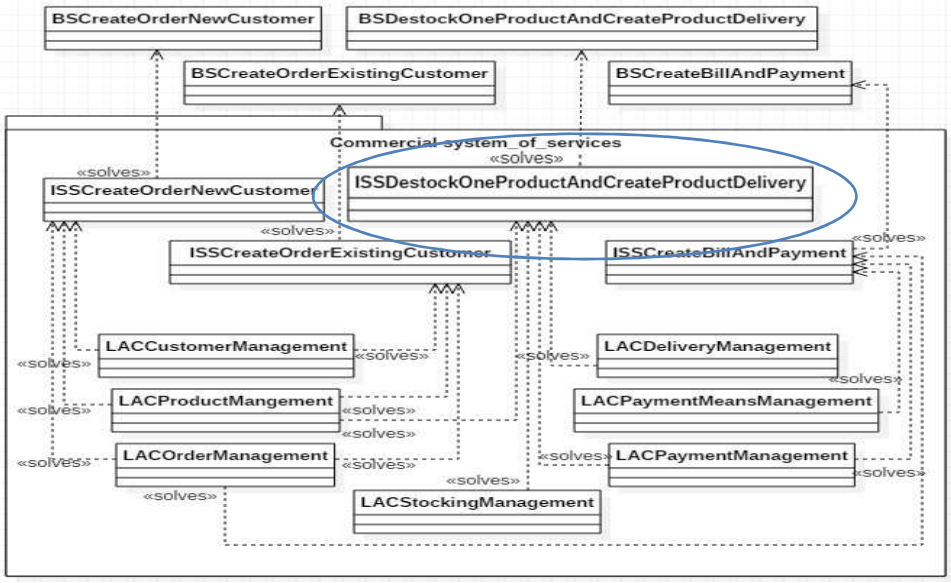

Fig. 20 – Illustration of the commercial system-of-services modification (enclosed) taking into account a delivery inside the store.

On the one hand, this evolution of the business preferences imply optimal coherence and coupling of the IS services conditioned by business services. On the other hand, the coherence and the coupling of the LACs conditioned by the IS services stay constant. This last result is due to the application of the ***System-of-Services-Logical-Architecture-Design*** pattern, which provides a robust (do not depend on business services change) definition of the LACs essential for coherence and of their dependencies at the origin of the coupling.

5.2. *Use case assessment*

According to the use case, a first benefit of the packaging system properties is to allow a measure of the improvement of the relevance of the business expert's specifications. The second benefit is due to the ***System-of-Services-Logical-Architecture-Design*** pattern,

which allows a high coupling of the LACs solving IS services and a low coherence of these LACs. The application of the pattern means a constant coherence and coupling of LACs, even if the IS services generation change after business services evolution. This constancy can be considered as proving robustness property of the logical architecture of a system-of-services based on the pattern.

The automatization of the generation of the IS services from business services indicates a quasi-real time for the logical architecture design and the physical architecture design, completed by architecture modeling and code automated generation, of the IS services. The IS services architect's intervention is reduced to the production of the alignment of the CIM with the TCMe, with the help of the business expert. However, this alignment requires first the design of TCMe (LACs and LACDs) of a system-of-services based on the pattern.

Besides, the proposed alternative designs have a significant cost. Indeed, this phase requires an expert reflection, first on the proposed architecture models, then exchanges with business experts in order to validate or not a solution. The prospect of automatizing the design of alternative scenarios in an enterprise requires the integration of business knowledge and logical architecture knowledge specific to the enterprise when generating such scenarios. The following discussion is about this design and the robustness property of the resulting logical architecture model.

6. Discussion

There is a gap between business specifications and the implementation (design of the physical architecture and generation of the associated code) of an IS service. The simplest development method is often to produce a textual description of the business service and extract a diagram of the data entities, including their attributes, participating in the service. The data entities are extracted because they are considered to be handled specifically in the service. The first illustration of ordering a product for a customer can thus induce two data entities: the product class that is selected from a sales catalog and the order class where the customer and the order reference are created when an order is placed. In a very simple way, the method would be to design the physical data model from these data entities and their association, then the physical application components managing these physical data, then the components managing the IS services and using the previous physical application components, and finally the interface from which the services are called by the user.

The first concern is the homogeneity of this service design, in the case where other services make it possible to order another type of product, for example. We can imagine in a second illustration that the customer is managed differently in the other application, because the company wants to make him commercial offers. In this case, the previously specified order data entity splits into a data entity dedicated only to the customer targeted by the marketing service, and another data entity targeting the order reference. On the other hand, it is also possible to imagine in a third illustration an order business service with a single participating data entity grouping the order, the customer and the product ordered, in the case, for example, of selling products in limited quantities and whose management is not a concern of the company. The problem of flexibility of the system-

of-services is thus raised because of the heterogeneity of services relating to the same business activity (order a product for a customer), but treated differently during the analysis, and therefore the implementation, of each IS service. This differentiated treatment obviously depends on the intent driving the analysis.

In order to ensure a homogeneous implementation, the design of a logical architecture common to all services of the system-of-services is proposed here. It automatically generates an implementation based on an alignment of business need (data entity attributes) with LACs. A point underlined for the use case is the necessary collaboration between the business expert and the logical architecture designer of IS services to achieve this alignment. As seen in this paper, the proposed pattern solves the problem of design granularity. The solution at the logical level is based on the lifetime of attributes to define consistent logical data. Stereotyping the LACs used in the pattern makes possible to identify the LACs representing an activity. These LACs manage a date on which an activity occurred or a reference associated with the activity. LACs with the stereotype "activity" are made dependent if in the company these activities follow one another (an activity depends on an activity if it precedes it in the activity diagram defining a business process). Similarly, "person", "document" or "reference" stereotypes allow tagging LACs have a longer lifespan than a LAC stereotyped "activity". A LAC stereotyped "activity" may thus depend on them within the system-of-services. The definition of these LACs is interesting for partitioning data at the logical level, such as for big data architecture,[65] and therefore its reliability.

These stereotypical LACs "activity" are in fact essential to ensure a low coupling between IS services. For example, a HMI developer of an ordering IS service will request an IS service allowing the selection of a one customer among the list of all customers. If this solution were chosen by the IS service architect, then it would increase the coupling of delivery, billing, or ordering IS services with this service in the commercial system-of-services, assuming they use this customer selection IS service. The logical architecture of this IS service is limited to the LAC of customer management (stereotyped "person" and not "activity"). In this case, the coupling properties at the IS services level are less efficient when it comes to defining a packaging system.

In order to define a packaging system, it therefore seems necessary to condition the logical architecture of each IS service through the presence of a stereotyped LAC "activity". In the example of the customer list, this means that the business expert can be interested in customers who have placed an order over a defined or indefinite period. In this case, the service of reading a customer list will become the service of reading a list of customers who have placed an order over a defined or indefinite period. If a filter on customer addresses is expected, then this service will include in its signature the definition of the filter on the customer's address. A solution is to overload the order reading IS service with these filtering parameters. The overloading of each IS service including at least one stereotyped LAC "activity" in its architecture reduces then the coupling at the IS service level as it reduces the total amount of IS services. This reinforces thus the packaging system properties of a system-of-services.

The multi-use of LACs designed with the ***System-of-Services-Logical-Architecture-Design*** pattern is based on the use of CRUD, which limits the useful functions in each LAC. This makes it easier to serendipitous reuse as proposed with the functions get, put, post and delete in the context of Web architecture.[66]

The reliability and reusability properties highlighted as consequence of the pattern use are interesting in the context of a sustainable system-of-services as defined in the EA objective.

However, in some cases, mathematical functions are added to the CRUD functions in a stereotyped "reference" LAC. This LAC, which can be assimilated to a mathematical library, is such that the others depend on it according to the ***System-of-Services-Logical-Architecture-Design*** pattern. This solution is interesting for technical functions such as, explicit or implicit, authentication functions, for example, in order to integrate them into the logic model of each IS service.

In addition, the reliability and reusability properties make it possible to design alternative solutions, sources of possible improvements to business services, IS services or logical architecture of the IS. The limit of the use of the algorithm and of these potential improvements is not only a logical design of the IS conforming to the business, but also an appropriation of this logical architecture by the designers and implementers of IS services who have to develop business requirements.

7. Conclusion

The question of the gap between business view point and system viewpoint of EA stays unavoidable when developing IS services of a system-of-services. Extending the concept of transformation contextual model enriching the PIM to the CIM, our work proposes an integration of a logical model of the system-of-services as context of the transformation of CIM into PIM. This integration needing the collaboration between business experts and IS service logical architects enable an automatization of the IS services generation and logical architecture, from the specification of business services (see ***Generation of Information System Services from Business Services*** algorithm). This IS services generation and logical architecture is completed by a more traditional automatized approach of IS services physical architecture modeling and coding.

The pattern enabling the design of a logical architecture model of a system-of-services, and the algorithm of IS services generation based on the logical architecture model of the system-of-services was tested on a use case. This test made it possible to highlight the properties of packaging system that can be improved following changes in business services that target an improvement in the business specification and not in the architecture of the system-of-services. It allowed also checking the packaging system properties at the LAC level thanks to the ***System-of-Services-Logical-Architecture-Design*** pattern.

Moreover reliability and partitioning properties of a packaging system based on the proposed pattern and algorithm have been underlined during the discussion in an EA approach. Future works encompass an extension to new use cases in order to propose other applications of the pattern, for example with robotic system.[67] Indeed, the logical model deduced from the pattern could be extended for an improved support of a business specification (management service or real-time service). In addition, more relevant measures of coupling and coherence within the packaging system will complete this perspective.

Appendix A. Implementation of Logical Application Component model checking

The code of the model transformation (lea2chk), which implements the checking of a logical application component model, is the following one. The meta-model's aggregating concept is called "LDA" (Logical Dynamic Architecture) in this implementation. It is used to describe the "lea" model of the checked logical enterprise architecture.

```
transformation lea2chk(source:lea);

main()
{
  log("Begin of the LAC model checking");
  log("\nOrientation of LACDs from life duration rule");
  source.rootObjects()[LDA]->checkLogicalPattern();
  log("\nLAC cycle checking");
  source.rootObjects()[LDA]->checkLogicalLoop();
  log("\nEnd of the LAC model checking");
}

query LogicalApplicationComponent::
  testFollowingComponents
  (seqComponents : Sequence(LogicalApplicationComponent)) :
  LogicalApplicationComponent
{
  var followingLogicalComponent :=
    self.lda.logicalApplicationComponent->select
(e : LogicalApplicationComponent |
e.stereotype=self.stereotype and
self.lda.logicalApplicationComponentDependency
    ->select(f :
LogicalApplicationComponentDependency |
seqComponents->select(g :
LogicalApplicationComponent |
g.name=f.source.name)->asSequence()->size()<>0
    and f.target.name=e.name)->asSequence()
->size()<>0)->asSequence();
  var test := true;
  var i := 1;
  while (i<=seqComponents->size() and test)
  {
    test := test and (seqComponents>at(i).name<>
  self.name);
```

```
    i := i + 1;
  };
  return(
    if test
    then
      if (followingLogicalComponent->size()<>0)
      then
        self.testFollowingComponents
    (followingLogicalComponent)
      else
        null
      endif
    else
      seqComponents->at(i - 1)
    endif);
}

query LDA::checkLogicalLoop()
{
  var i := 1;
  while (i<=self.logicalApplicationComponent->size())
  {
    var followingComponents :=
      self.logicalApplicationComponent->
      select(e : LogicalApplicationComponent  |
      e.stereotype = self.logicalApplicationComponent->at(i).stereotype and
      self.logicalApplicationComponentDependency->
      select(f : LogicalApplicationComponentDependency |
      self.logicalApplicationComponent->at(i).name=f.source.name and
      f.target.name=e.name)->asSequence()->size()<>0)->asSequence();
    if (followingComponents->size()<>0)
    then
      if (self.logicalApplicationComponent->
        at(i).testFollowingComponents(followingComponents)<>null)
      then
        log("WARNING: there is a logical loop from " +
          self.logicalApplicationComponent->at(i).name + " to itself")
      endif
    endif;
    i := i + 1;
  };
}
```

```
query LDA::checkLogicalPattern()
  {
    var seqfc := self.logicalApplicationComponent->
      select(e : LogicalApplicationComponent | e.stereotype<>"activity" and
      e.stereotype<>"person" and e.stereotype<>"document" and
      e.stereotype<>"reference")->asSequence();
    if (seqfc->size()=0)
    then
    {
      var seqdep := self.logicalApplicationComponentDependency;
      var i := 1;
      while (i<=seqdep->size())
      {
        if(((seqdep->at(i).source.stereotype="document" or
          seqdep>at(i).source.stereotype="person" or
          seqdep->at(i).source.stereotype="reference")
          and seqdep->at(i).target.stereotype="activity") or
          (seqdep->at(i).source.stereotype="person" and
          seqdep->at(i).target.stereotype="document") or
          (seqdep->at(i).source.stereotype="reference" and
          (seqdep->at(i).target.stereotype="person" or
          seqdep->at(i).target.stereotype="document" or
          seqdep->at(i).target.stereotype="activity")))
        then
          log("WARNING: the dependency from " +
            seqdep->at(i).source.name + " to " +
            seqdep->at(i).target.name + " does not satisfy the pattern : " +
            seqdep->at(i).target.stereotype + " to " +
            seqdep->at(i).source.stereotype)
        endif;
        i := i + 1;
      };
    }
    else
    {
      log("WARNING: the stereotypes of the LACs must be \"activity\" or \"person\"
        or \"document\" or \"reference\" => the pattern cannot be checked");
      var j := 1;
      while (j<=seqfc->size())
      {
        log("    - " + seqfc->at(j).name + " is stereotyped \"" + seqfc->at(j).stereotype +
```

```
            "\"");
          j := j + 1;
        };
      }
      endif;
    }
```

Appendix B. Automatic generation of physical architecture design and code generation for commercial system-of-services use case

The sequence of physical operations encapsulated in the loop combined fragment contains:

- The *ISSReadAProduct:readProduct_name* IS service call with the *BOReadAProduct:readProduct_name* business operation;
- The *DOcreateProduct_name* data operation carrying out the persistence of the created data representing the product provided by *BOReadAProduct:readProduct_name*;
- The *DOcreateJointOrderProduct* data operation carrying out the persistence of the created joint of the product data and the order data previously (created just before the loop combined fragment by the *DOcreateOrder* data operation).

The joint of the product data and the order data is also designed in the physical data model in Fig. 21.

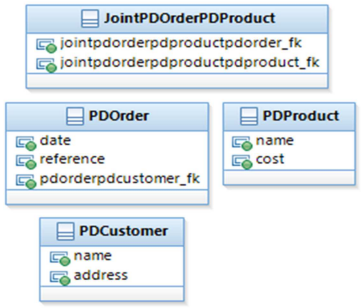

Fig. 21 – *ISSCreateOrderExistingCustomer:createOrder_date* IS service static physical data model (excerpted from the physical data model) transformed into UML2 class diagram.

PDM contains the rules enabling the generation of a relational database, especially those about the foreign key management:

- *pdorderpdcustomer_fk*, from *PDOrder* physical data to *PDCustomer* physical data;
- *jointpdorderpdproductpdorder_fk* from *JointPDOrderPDProduct* to *PDOrder*;
- *jointpdorderpdproductpdproduct_fk* from *JointPDOrderPDProduct* to *PDProduct*.

This excerpt of the database is generated from the SQL script automatically generated by a model transformation. This transformation implements some rules constraining the sequence of the table creations, and of the prior tables cleaning.

The generation of Java code of the IS service on the business layer is in Fig. 22. The code is simplified because it is not included into the contribution of this paper. However, the relevancy of this code compared to the physical components sequence diagram (see description above) is noteworthy.

```
@Override
public   void   BOCreateOrderExistingCustomer:createOrder_date(String   nameCustomer,   String
addressCustomer, String dateOrder,
int referenceOrder, String nameProduct, String costProduct)
{
     /**
      * Exception if not: <>null
      */
     customerDAO.DOreadCustomer_name(customer);
     /**
      * Exception if not: <>null
      */
     orderDAO.DOcreateOrder_date(order);
     /**
      * Loop start
      */
     Customer customer =
          this.BOReadAProduct:readProduct_name(nameCustomer,
               addressCustomer)
          productDAO.DOcreateProduct_name(product);
          jointorderproductDAO.DOcreateJointOrderProduct
               (jointorderproduct);
     /**
      * Loop end
      */
}
```

Fig. 22 – *ISSCreateOrderExistingCustomer:createOrder_date* IS service simplified code (business layer).

References

1. K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay and M. Munro, Service-based software: the future for flexible software, in *Proceedings Seventh IEEE Asia-Pacific Software Engeering Conference (APSEC)* (2000), pp. 214–221.
2. D. Sprott and L. Wilkes, Understanding service-oriented architecture, *The Architecture Journal* **1**(1) (2004), pp.10–17.
3. J. A. Zachman, The Zachman framework for enterprise architecture: primer for enterprise engineering and manufacturing, *Zachman International* (2003).
4. A. Alwadain, E. Fielt, A. Korthaus and M. Rosemann, Where do we find services in enterprise architectures? A comparative approach, in *Proceedings of the 22nd Australasian Conference on Information Systems (ACIS)* (2011).
5. H. M. Chen, Towards service engineering: service orientation and business-IT alignment, in *Proceedings of the 41st IEEE Annual Hawaii International Conference on System Sciences (HICSS)* (2008), pp. 114–114.
6. H. Jonkers, M. Lankhorst, R. Van Buuren, S. Hoppenbrouwers, M. Bonsangue and L. Van Der Torre, Concepts for modeling enterprise architectures, *International Journal of Cooperative Information Systems* **13**(03) (2004), pp. 257–287.
7. The Open Group, The TOGAF® Standard Version 9.2. [Accessed 10 04 2019] (2018), http://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html.
8. S. M. Glissmann and J. Sanz, An approach to building effective enterprise architectures, in *Proceedings of the 44th IEEE Hawaii International Conference on System Sciences* (2001), pp. 1–10.

9. B. H. Cameron and E. McMillan, Analyzing the current trends in enterprise architecture frameworks, in *Proceeding of Journal of Enterprise Architecture* **9**(1) (2013), pp. 60-71.
10. A. Dongre, Data quality and integrity management for telecom operators, *Telecom Business Review* **7**(1) 2014, pp. 1–8.
11. B. Michelberger, B. Mutschler and M. Reichert, Process-oriented information logistics: Aligning enterprise information with business processes, in *Proceedings of the IEEE 16th International Enterprise Distributed Object Computing (EDOC)* (2012), pp. 21–30.
12. O. El-Telbany and A. Elragal, Business-information systems strategies: a focus on misalignment, *Procedia Technology* **16** (2014), pp. 250–262.
13. D. Yue, L. Wanjun, L. Cuicui, F. Wenxiang, Based on SOA architecture and component software reuse architecture research, in *Proceedings of the 2nd IEEE International Conference on Information Management and Engineering* (2010), pp. 517–520.
14. C. Erbas and B. C. Erbas, On a theory of software engineering a proposal based on transaction cost economics, in *Proceedings of the 2nd IEEE SEMAT Workshop on a General Theory of Software Engineering (GTSE)* (2013), pp. 15–18.
15. B. Van Gils, Strategy and architecture–reconciling worldviews, in Proceedings of the Working Conference on Practice-Driven Research on Enterprise Transformation (2009), pp. 181–196.
16. T. Erl, *SOA Principles of Service Design*, (Prentice Hall, 2007).
17. N. Bieberstein, S. Bose, L. Walker and A. Lynch, Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals, *IBM systems journal* **44**(4) (2005), pp. 691–708.
18. R. Perrey and M. Lycett, Service-oriented architecture, in *Proceedings of IEEE Symposium on Applications and the Internet Workshops* (2003), pp. 116–119.
19. N. Joachim, D. Beimborn, F. Schlosser and T. Weitzel, Does SOA create or require IT/business collaboration? Investigating SOA's potential to reduce the gap between IT and business, *32nd International Conference on Information Systems (ICIS)* (2011).
20. B. Molnár and A. Tarcsi, Architecture and system design issues of contemporary web-based information systems, in Proceedings of the 5th International IEEE Conference on Software, Knowledge Information, Industrial Management and Applications (SKIMA) (2011), pp. 1–8.
21. S. Kotusev, Enterprise architecture: what did we study?, *International Journal of Cooperative Information Systems*, **26**(04) (2017), pp. 1730002-1–1730002-84.
22. M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, **17**(2) (2008), pp. 223–255.
23. D. Ameller, X. Burgués, O. Collell, D. Costal, X. Franch and M. P. Papazoglou, Development of service-oriented architectures using model-driven development: A mapping study. *Information and Software Technology*, **62** (2015), pp. 42–66.
24. M. López-Sanz and E. Marcos, ArchiMeDeS: A model-driven framework for the specification of service-oriented architectures. *Information Systems*, **37**(3) (2012), pp. 257-268.
25. J. Simonin and J. Puentes, Automatized integration of a contextual model into a process with data variability, *Computer Languages, Systems & Structures* **54** (2018), pp. 156–182.
26. I. Todoran, Z. Hussain and N. Gromov, SOA integration modeling: An evaluation of how SoaML completes UML modeling, in *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference Workshops* (2011), pp. 57–66.
27. A. M. Olson, R. R. Raje, B. Devaraju and L. S. Gallege, Learning improves service discovery, *Concurrency and Computation: Practice and Experience* **27**(7) (2015), pp. 1679–1694.
28. R. Paul, W. T. Tsai and J. Bayne, The impact of SOA policy-based computing on C2 interoperation and computing, *10th international command and control research and technology symposium (ICCRTS)* (2005).
29. A. Zimmermann, K. Sandkuhl, M. Pretz, M. Falkenthal, D. Jugel and M. Wissotzki, Towards an integrated service-oriented reference enterprise architecture, in *Proceedings of the 2013 ACM International Workshop on Ecosystem Architectures* (2013), pp. 26–30.

30. S. Bondar, J. C. Hsu, A. Pfouga and J. Stjepandić, Agile digital transformation of System-of-Systems architecture models using Zachman framework, *Journal of Industrial Information Integration* **7** (2017), pp. 33–43.
31. S. Majd, M. H. Abel and M. Alok, An architectural model for system of information systems, in *Proceedings of the OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"* (2015), pp. 411–420.
32. P. Salvaneschi, Modeling of information systems as systems of systems through DSM, in Proceedings of the 4th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems (SESoS) (2016), pp. 8–11.
33. J. Xiong, B. F. Ge, X. K. Zhang, K. W. Yang and Y. W. Chen, Evaluation method of system-of-systems architecture using knowledge-based executable model, in *Proceedings of 17th IEEE Annual Conference International Conference on Management Science & Engineering* (2010), pp. 141–147.
34. T. Bianchi, D. S. Santos and K. R. Felizardo, Quality attributes of systems-of-systems: A systematic literature review, in *Proceedings of 3rd IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems* (2015), pp. 23–30.
35. J. S. Topper and N. C. Horner, Model-based systems engineering in support of complex systems development, *Johns Hopkins APL technical digest* **32**(1) (2013).
36. Z. Stojanovic, A. Dahanayake and H. Sol, Modeling and design of service-oriented architecture, in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics* **5** (2004), pp. 4147–4152.
37. M. Rosemann, P. Green, M. Indulska and J. C. Recker, Using ontology for the representational analysis of process modelling techniques, *International Journal of Business Process Integration and Management* **4**(4) (2009), pp. 251–265.
38. R. Kazman, K. Schmid, C. B. Nielsen and J. Klein, Understanding patterns for system of systems integration, in *Proceedings of the 8th IEEE International Conference on System of Systems Engineering* (2013), pp. 141–146.
39. J. Klein and H. Van Vliet, A systematic review of system-of-systems architecture research, in *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures* (2013), pp. 13–22.
40. I. G. Vargas, T. Gottardi and R. T. V. Braga, Approaches for integration in system of systems: a systematic review, in *Proceedings of the 4th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems (SESoS)* (2016), pp. 32–38.
41. B. Elvesæter, D. Panfilenko, S. Jacobi and C. Hahn, Aligning business and IT models in service-oriented architectures using BPMN and SoaML. In *Proceedings of the First International Workshop on Model-Driven Interoperability* (2010), pp. 61–68.
42. A. Delgado, F. Ruiz, I. G. R. de Guzmán and M. Piattini, Model transformations for Business-IT alignment: from collaborative business process to SoaML service model. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (2012), pp. 1720-1722.
43. C. A. Whitcomb, M. Auguston and K. Giammarco, *Composition of Behavior Models for Systems Architecture* **14** (John Wiley & Sons 2015), pp. 361–391.
44. N. Kulkarni and V. Dwivedi, The role of service granularity in a successful SOA realization a case study, in *Proceedings of the IEEE Congress on Services* **I** (2008), pp. 423–430.
45. K. J. Sullivan, W. G. Griswold, Y. Cai and B. Hallen, The structure and value of modularity in software design, in *Proceedings of ACM SIGSOFT Software Engineering Notes* **26**(5) (2001), pp. 99–108.
46. C. Pahl, P. Jamshidi and O. Zimmermann, Architectural principles for cloud software, *ACM Transactions on Internet Technology (TOIT)* **18**(2) (2018) 17.
47. A. K. Raz, C. R. Kenley and D. A. DeLaurentis, A System-of-Systems perspective for information fusion system design and evaluation, *Information Fusion* **35** (2017), pp. 148–165.
48. B. Solaiman, É Bossé, L. Pigeon, D. Guériot and M. C. Florea, A conceptual definition of a holonic processing framework to support the design of information fusion systems, *Information Fusion* **21** (2015), pp. 85–99.

49. A. Giret, E. Garcia and V. Botti, An engineering framework for service-oriented intelligent manufacturing systems, *Computers in Industry* **81** (2016), pp. 116–127.
50. R. J. Cloutier and D. Verma, Applying the concept of patterns to systems architecture. *Systems engineering* **10**(2) (2007), pp. 138–154.
51. S. Cook, Looking back at UML, *Software & Systems Modeling* **11**(4) (2012), pp. 471–480.
52. E. C. Ferstl and D. Y. von Cramon, The role of coherence and cohesion in text comprehension: an event-related fMRI study, *Cognitive Brain Research* **11**(3) (2001), pp. 325–340.
53. A. Dillon, *User acceptance of information technology* (Taylor and Francis, London, 2001).
54. M.J. Shepperd and D. Ince, *Derivation and Validation of Software Metrics* (Oxford University Press, 1993.
55. D. M. Eriksson, A principal exposition of Jean-Louis Le Moigne's systemic theory, *Cybernetics & Human Knowing* **4**(2–3) (1997).
56. G. Gui and P. D. Scott, Coupling and cohesion measures for evaluation of component reusability, in *Proceedings of the ACM international workshop on Mining software repositories* (2006), pp. 18–21.
57. K. M. Hansen and K. Manikas, (Automated) software modularization using community detection, in *Proceedings of European Conference on Software Architecture* (2015), pp. 95-102.
58. Object Management Group, UML 2.2 Unified Modeling Language [Accessed 04 30 2019] (2009), http://www.omg.org/spec/UML/2.2/.
59. M. Hammer and J. Champy, *Reengineering the Corporation* (Harper Collins, New York, 1993).
60. S. W. Ambler, Process patterns: building large-scale systems using object technology, (Cambridge university press, 1998).
61. C. Alexander, *The timeless way of building* (Oxford University Press, New York, 1979).
62. M. Hagen and V. Gruhn, Towards flexible software processes by using process patterns, in *Proceedings of the IASTED Conference on Software Engineering and Applications* (2004), pp. 436–441.
63. Object Management Group, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification [Accessed 10 04 2019] (2016), http://www.omg.org/spec/QVT/1.3.
64. J. Simonin, Method of modelling reference data and use of this method for localization of reference data in an information system, U.S. Patent No. 7,249,134 (2007).
65. H. Salavati, T. J. Gandomani and R. Sadeghi, A robust software architecture based on distributed systems in big data healthcare, *in Proceedings of the IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (2017), pp. 1701–1705.
66. S. Vinoski, Serendipitous Reuse, *IEEE Internet Computing* **12**(1) (2008), pp. 84–87.
67. A. Ahmad and M. A. Babar, Software architectures for robotic systems: A systematic mapping study, *Journal of Systems and Software* **122** (2016), p. 16–39.