

Aspect-Oriented Schema Evolution in Object Databases: A Comparative Case Study

Awais Rashid

Computing Department, Lancaster University, Lancaster LA1 4YR, UK

awais@comp.lancs.ac.uk

Abstract.

This paper provides a comparative evaluation of two aspect-oriented features of the SADES object database evolution system with three other systems each representing a particular category of evolution systems. The evaluation is based on a case study involving a design correction scenario. The features compared include schema relationships and instance adaptation. The discussion demonstrates the cost-effectiveness and resilience of an aspect-oriented approach in the face of unanticipated changes, customisations and extensions.

1. Introduction

The schema of a database is subject to change due to a variety of reasons e.g. the need to correct mistakes in the database design, add new features during incremental design or reflect changes in the structure of the real world artefacts modelled in the database. Consequently, like all database systems, schema evolution and corresponding adaptation of instances (data) and applications is a critical requirement in an object database system. Previously we have argued that schema evolution mechanisms should be adaptable in the face of changes in organisational needs and processes, specialised “local” requirements and volatile business domains [15] [16] [17]. We have mainly focused on object database systems in this context as customisation is a challenging problem in the presence of a rich data model (as compared to relational systems) and inherent support for complex applications such as computer-aided design and manufacturing. Our solutions have been based on the use of aspect-oriented programming (AOP) techniques [5] for the separation of crosscutting, customisable features such as instance adaptation (simulated or physical conversion of objects in accordance with schema changes), versioning, links among persistent entities, change propagation and referential integrity semantics [10] [12]. AOP makes it possible to separate these crosscutting concerns through abstractions known as aspects and provides mechanisms, known as weaving, to compose the aspects together with entities they cut across. The aspect-oriented approach has been employed to provide cost-effective, localised changes during both schema evolution and customisation in the SADES object database evolution system [9] [10].

This paper uses a case study to provide a comparative evaluation of some of the aspect-oriented features of SADES with three other object database evolution systems: ORION [2], ENCORE [18] and TSE [8]. The case study is aimed at demonstrating the cost-effectiveness and customisability of changes in SADES. The three systems have been chosen because each system:

- offers a set of evolution primitives comparable or closely comparable to those offered by SADES.
- represents a particular category of evolution systems. ORION represents the *schema modification* category where the database has one logical schema to which all changes are applied¹. ENCORE represents the *class versioning* category where a new version of a class is created upon modification and instances are bound to specific class versions. TSE represents the *schema versioning* category which allows several versions of one logical schema to be created and manipulated independently².
- facilitates dynamic schema evolution as is the case with SADES.

Note that no commercial systems have been chosen for comparison. Our earlier evaluation of schema evolution in commercial systems [14] identified a lack of support for addition, deletion and manipulation of non-leaf classes. As discussed later, this is a required feature in the context of the case study.

The next section describes the evolution scenario for the case study. Section 3 describes the realisation of the scenario in each of the four systems. Section 4 compares two aspect-oriented features of SADES: schema relationships and instance adaptation with the other three systems on the basis of the evolution scenario realisation in section 3. It should be noted that a detailed description of SADES is beyond the scope of this paper. Only relevant features of SADES are discussed during the comparison. Interested readers are referred to [9] [10] [13] [15] [16] [17] for further details of the system. Section 5 concludes the paper and identifies directions for future work.

¹ Another version of Orion supports versioning of schemas. However, for the purpose of this discussion only the basic *schema modification* system is considered.

² SADES itself acts as a representative of the *hybrid category* as it superimposes *schema modification* on *class versioning*. However, the aim of this case study is not evolution model comparison. This will form the subject of a separate paper.

2. Evolution Scenario

The case study presented in this section was carried out at an adult education organisation where the day-to-day activities revolve around the database which stores information about students, staff, courses, examination and results, etc. The organisation management decided to redesign the database due to a variety of reasons:

- Structural changes in the organisation and courses were to be reflected in the database.
- Design mistakes in the old database needed to be corrected.
- The database structure had become cluttered over the years due to limited support for evolution.
- Major structural changes and design corrections were not possible in the absence of good support for evolution.

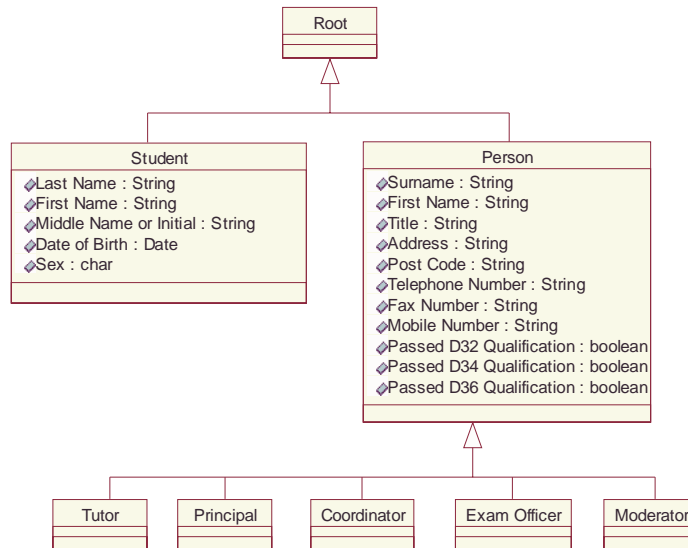


Fig. 1: Database Structure before Evolution

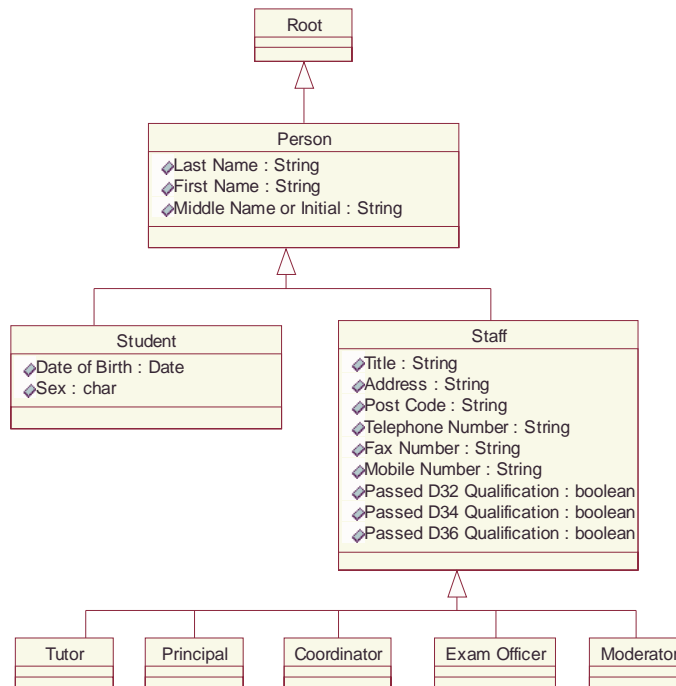


Fig. 2: Database Structure after Evolution

The evolution scenario chosen for the discussion in this paper is one involving design correction. It is assumed that the class hierarchy is single-rooted. Fig. 1 shows the database structure prior to evolution. The class hierarchy does not conform to good OO design principles as the class *Student* is not a subclass of the class *Person*. Fig. 2 shows the structure to be adopted to conform to good practice. In this structure the class *Student* becomes a subclass of the class *Person*. The class *Person* defines attributes common to both student and staff objects. A non-leaf class *Staff* is introduced to capture attributes specific to staff objects.

A general sequence of evolution operations to be carried out in each system in the case study is as follows:

1. In class *Person* rename the attribute *Surname* to *Last Name*, drop the attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification* and add an attribute *Middle Name or Initial*.
2. Create a new non-leaf subclass of *Person* defining attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification*. Note that a more useful operation will be the one that *moves* attributes from *Person* to *Staff* in an information preserving fashion. This will be discussed as an example customisation in section 4 as none of the four systems supports such a primitive.
3. Drop attributes *Last Name*, *First Name* and *Middle Name or Initial* from *Student*.
4. Reposition the class *Student* as a leaf subclass of *Person*.

3. Scenario Realisation in Chosen Systems

This section presents a realisation of the evolution scenario in each of the four systems. This discussion is important as it acts as a basis (in section 4) to compare the impact of schema changes and customisations in SADES and Orion, ENCORE and TSE.

3.1 Orion

The evolution scenario involves introduction of a non-leaf class *Staff* and repositioning the class *Student* in the hierarchy graph. These operations are not supported by ORION. It is not possible to simulate these operations through a sequence of addition and deletion of edges in the class hierarchy graph because the system is single-rooted and isolated graph nodes are not allowed. Consequently it will not be possible to realise this simple evolution scenario. However, for the purpose of this analysis it is assumed that introduction of non-leaf classes and repositioning of classes is possible through a sequence of addition and deletion of hierarchy graph edges. Based on this assumption the sequence of evolution operations in ORION will be as shown in fig. 3. The conceptual structure of the database after evolution will be the same as shown in fig. 2.

1. Drop the hierarchy graph edge between *Student* and *Root*.
2. Add a hierarchy graph edge from *Student* to *Person*.
3. Drop the attributes *Last Name*, *First Name* and *Middle Name or Initial* from *Student*.
4. Drop the attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification* from *Person*.
5. Rename the attribute *Surname* in *Person* to *Last Name*.
6. Add a new attribute *Middle Name or Initial* to *Person*.
7. Introduce a new leaf subclass *Staff* of *Person* with attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification*.
8. Drop the hierarchy graph edges from *Tutor*, *Principal*, *Coordinator*, *Exam Officer* and *Moderator* to *Person*.
9. Add hierarchy graph edges from *Tutor*, *Principal*, *Coordinator*, *Exam Officer* and *Moderator* to *Staff*.

Fig. 3: Sequence of evolution operations in ORION

3.2 ENCORE

ENCORE offers the necessary primitives to perform the required evolution. The sequence of evolution operations in ENCORE will be as shown in fig. 4. The conceptual structure of the database after evolution in ENCORE is shown in fig. 5. Note that ENCORE does not facilitate removal of redundant, intermediate class versions automatically generated by the system. These versions are only significant during evolution and include *Tutor_V2*, *Principal_V2*, *Coordinator_V2*, *Exam Officer_V2*, *Moderator_V2* and *Student_V2*.

1. Create a new version (*Person_V2*) of *Person* by dropping the attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification*, adding the attribute *Middle Name or Initial* and renaming the attribute *Surname* to *Last Name*. This will automatically generate new versions of the subclasses of *Person*: *Tutor*, *Principal*, *Coordinator*, *Exam Officer* and *Moderator*.
2. Introduce an initial version (*Staff_V1*) for *Staff*. The new class and hence its initial version forms a non-leaf node in the hierarchy graph and defines attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification*. *Staff_V1* will inherit from *Person_V2*. New versions will automatically be derived from existing subclass versions of *Person_V2*: *Tutor_V2*, *Principal_V2*, *Coordinator_V2*, *Exam Officer_V2* and *Moderator_V2*. The new subclass versions *Tutor_V3*, *Principal_V3*, *Coordinator_V3*, *Exam Officer_V3* and *Moderator_V3* will inherit from *Staff_V1*.
3. Create a new version (*Student_V2*) of *Student* by dropping the attributes *Last Name*, *First Name* and *Middle Name or Initial*.
4. Create a new version (*Student_V3*) of *Student* having the new version of *Person* (*Person_V2*) as its superclass version.

Fig. 4: Sequence of evolution operations in ENCORE

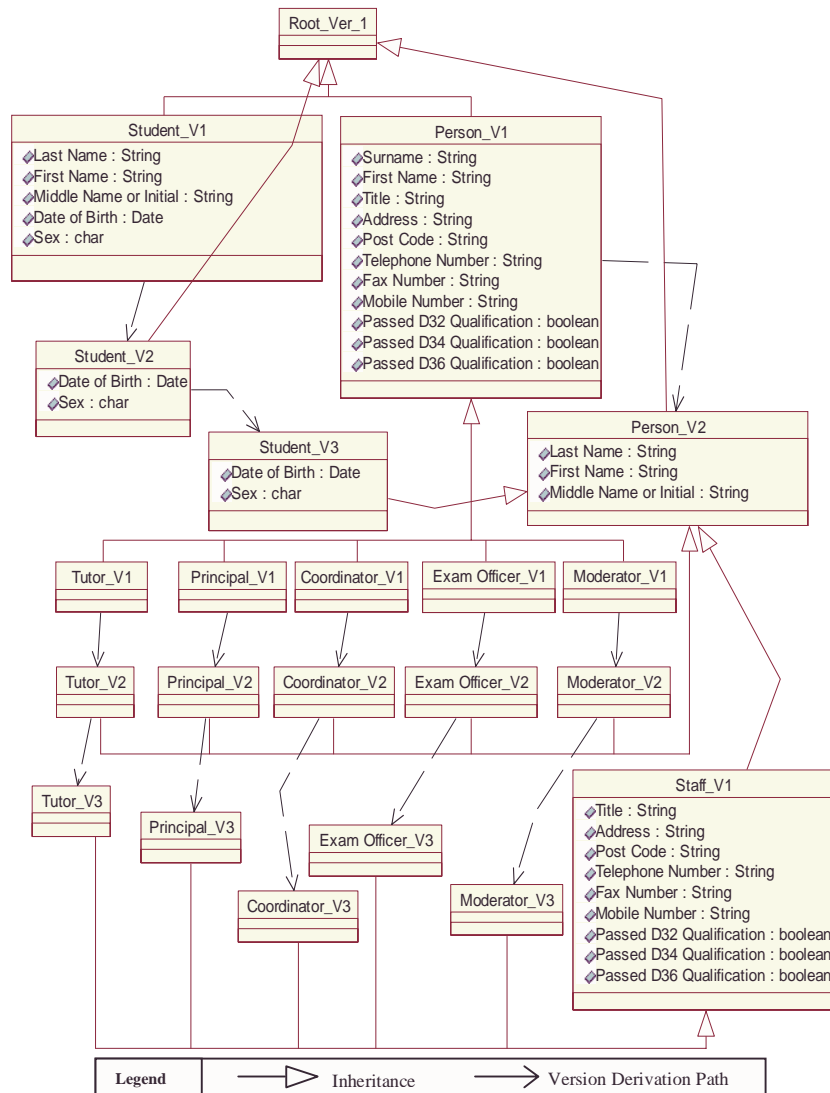


Fig. 5: Database Structure after Evolution in ENCORE

3.3 TSE

Like ORION, TSE does not support the addition of non-leaf classes or the repositioning of classes. For the purpose of this analysis it is assumed that these operations are possible through a sequence of addition and deletion of hierarchy graph edges. Based on this assumption the sequence of evolution operations in TSE will be as shown in fig. 6. Note that TSE employs *virtual* classes during evolution. A virtual class is the same as any other class. It is called *virtual* in TSE because it is introduced specifically for evolution.

1. Create a virtual class *Person'* to be inherited by the class *Person*. *Person'* will only define the attributes *Last Name* and *First Name*. Note that in TSE renaming is carried out at the view level and a mapping is maintained between names in views and the global schema. The introduction of *Person'* will result in automatic introduction of virtual superclasses for all the subclasses of *Person* i.e. *Tutor'*, *Principal'*, *Coordinator'*, *Exam Officer'* and *Moderator'* will be created.
2. Create a virtual class *Person''* inheriting from *Person'* and defining the attribute *Middle Name or Initial*. The creation of *Person''* will result in automatic introduction of virtual subclasses for all the subclasses of *Person'* i.e. *Tutor''*, *Principal''*, *Coordinator''*, *Exam Officer''* and *Moderator''* will be created.
3. Create a new leaf class *Staff* inheriting from *Person''* and defining attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification*.
4. In order to introduce *Staff* as a non-leaf subclass of *Person''* create virtual classes *Tutor'''*, *Principal'''*, *Coordinator'''*, *Exam Officer'''* and *Moderator'''* which will inherit from a newly created virtual subclass of *Staff*: *Staff'*.
5. Create a virtual superclass (*Student'*) of *Student* defining the attributes *Date of Birth* and *Sex*.
6. In order to make *Student'* a subclass of *Person''* introduce a virtual class *Student''* inheriting from both *Student'* and *Person''*.

Fig. 6: Sequence of evolution operations in TSE

The conceptual structure of the database after evolution in TSE is shown in fig. 4. In TSE schema versioning is simulated using views. The bold line in fig. 7 represents the view to be selected using the view specification language.

3.4 SADES

SADES offers primitives for addition of non-leaf classes and repositioning existing classes in the system. Therefore, the evolution scenario can be realised without problems. The sequence of evolution operations in SADES will be as shown in fig. 8.

The conceptual structure of the database after evolution in SADES is shown in fig. 9. Note that SADES superimposes *schema modification* on *class versioning* in order to provide maintainers with a coherent and comprehensible view of the system in a fashion similar to *schema modification* while maintaining change histories at a fine granularity in the same manner as *class versioning*. It should also be noted that SADES also offers features to remove redundant class versions generated for evolution only. These can be removed without loss of information or consistency in SADES. However, for the purpose of this case study this feature has been ignored in order to provide a fair comparison with ENCORE.

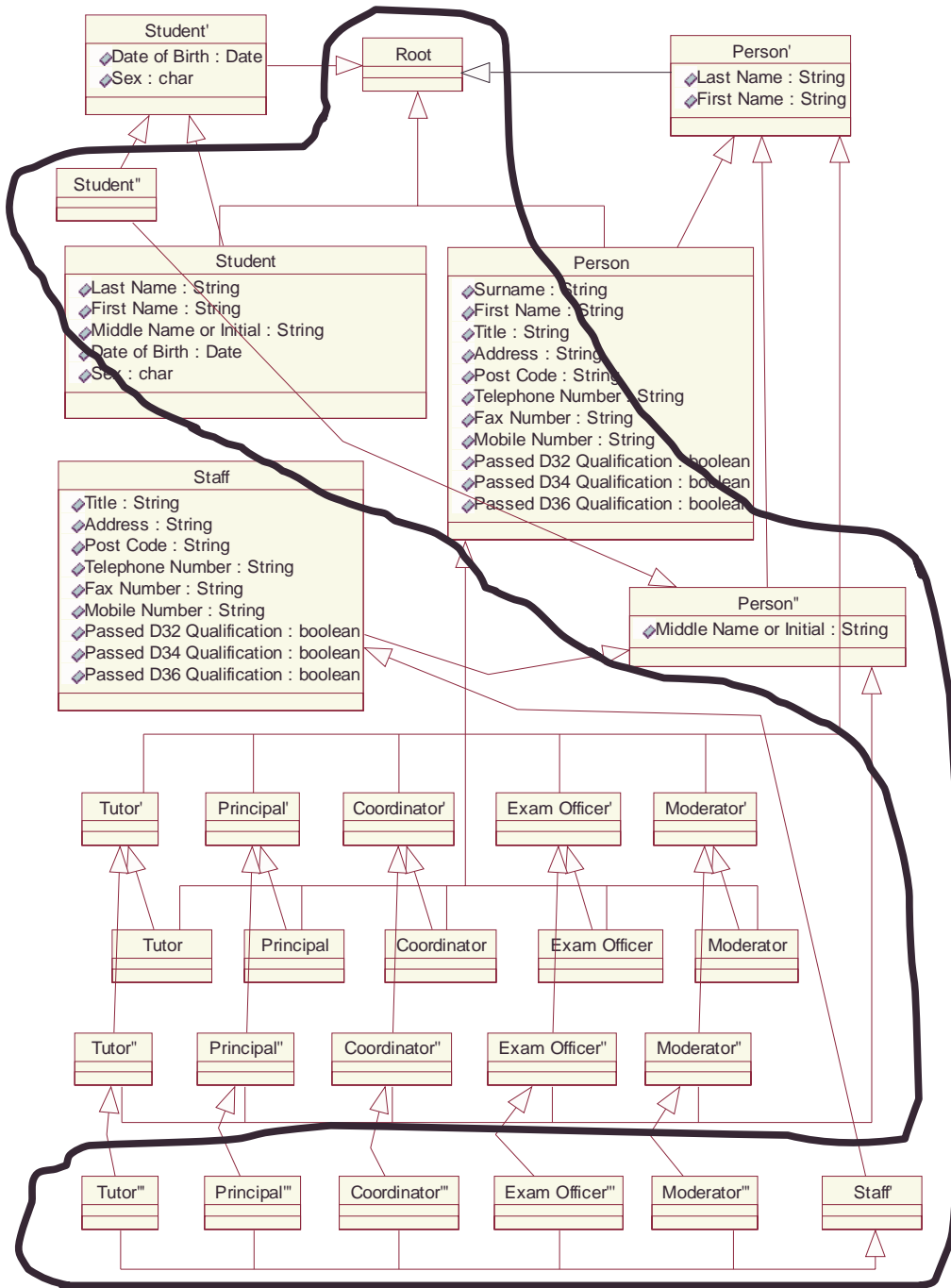


Fig. 7: Database Structure after Evolution in TSE

1. In class *Person* rename the attribute *Surname* to *Last Name*, drop the attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification* and add an attribute *Middle Name or Initial*. This will automatically create a new version *Person_V2* of class *Person*. New versions for existing subclasses of *Person* will also be created. The new subclass versions will inherit from *Person_V2*.
2. Create a new non-leaf subclass of *Person* defining attributes *Title*, *Address*, *Post Code*, *Telephone Number*, *Fax Number*, *Mobile Number*, *Passed D32 Qualification*, *Passed D34 Qualification* and *Passed D36 Qualification*. An initial version *Staff_V1* defining the specified attributes will be created by the system. *Staff_V1* will inherit from the default version for *Person* (*Person_V2*). New versions of the classes: *Tutor*, *Principal*, *Coordinator*, *Exam Officer* and *Moderator* will be introduced by the system. The new versions will inherit from the initial version for *Staff* (*Staff_V1*). Stubs will be generated for the class version derivation graph (CVDG) for *Staff* in order to maintain links between class versions of *Person* and their older subclass versions. These stubs will be invisible to the maintainer who will perceive a direct CVDG-level inheritance relationship between versions of *Person* and their older subclass versions.
3. Drop attributes *Last Name*, *First Name* and *Middle Name or Initial* from *Student*. This will automatically create a new version of *Student* (*Student_V2*).
4. Reposition the class *Student* as a leaf subclass of *Person*. This will create a new version *Student_V3* of *Student* inheriting from the default version for *Person* (*Person_V2*).

Fig. 8: Sequence of evolution operations in SADES

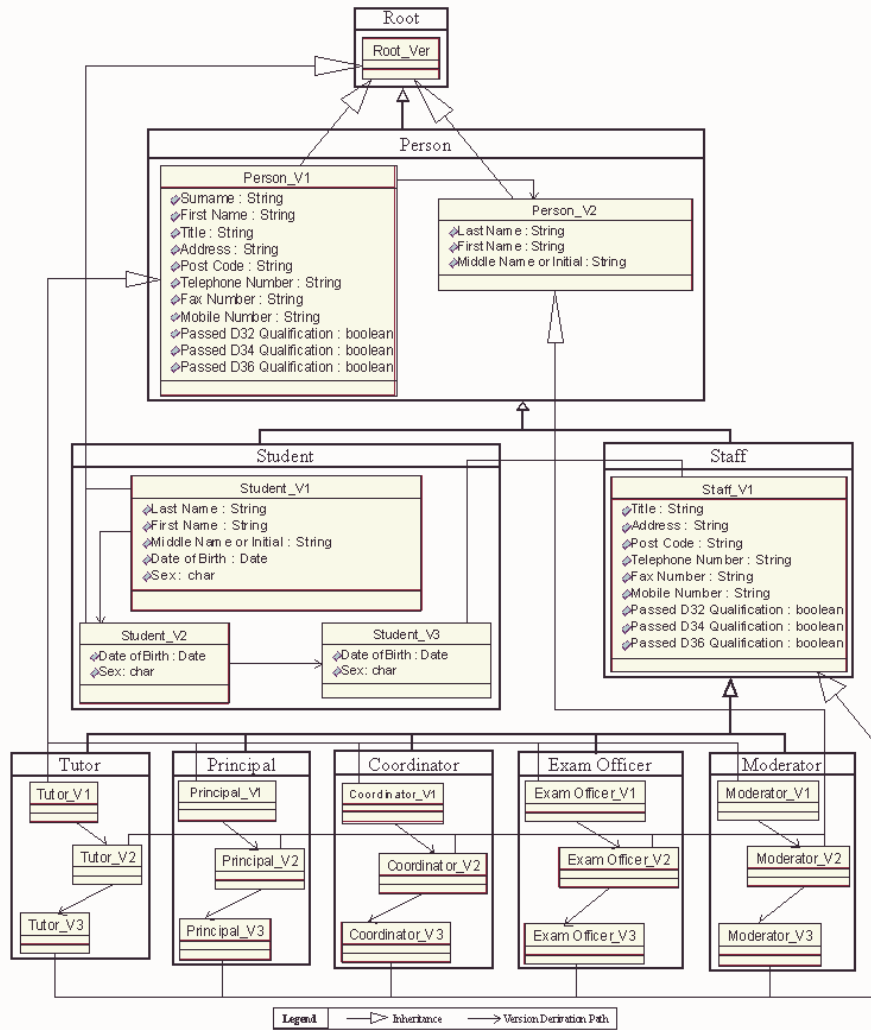


Fig. 9: Database Structure after Evolution in SADES

4. Comparative study

4.1 Schema Relationships

Meta-objects are the elements that form the schema in an object database. These include objects representing class, attribute and method definitions. Meta-objects are instances of meta-classes which define structure and behaviour of meta-objects. ORION, ENCORE and TSE use attributes at the meta-object level to implement relationships among meta-objects. Examples of these relationships are inheritance relationships among *class* meta-objects and aggregation relationships between a *class* meta-object and *attribute* and *method* meta-objects. The implementation of inheritance relationships in these systems is shown in fig. 10(a). For simplicity only the structure in Orion (a *schema modification* system) is shown. A similar structure is employed by ENCORE and TSE (in the context of their particular evolution models). Each *class* meta-object maintains collections of references to *class* meta-objects forming its superclasses and subclasses. Such a structure introduces the evolution problem at the meta-object level. When any relationships are modified corresponding attributes in the affected meta-objects need to be updated to reflect the change. Modifying the structure of meta-classes and introduction/removal of existing classes is very expensive. This reduces the extensibility and maintainability of the evolution framework.

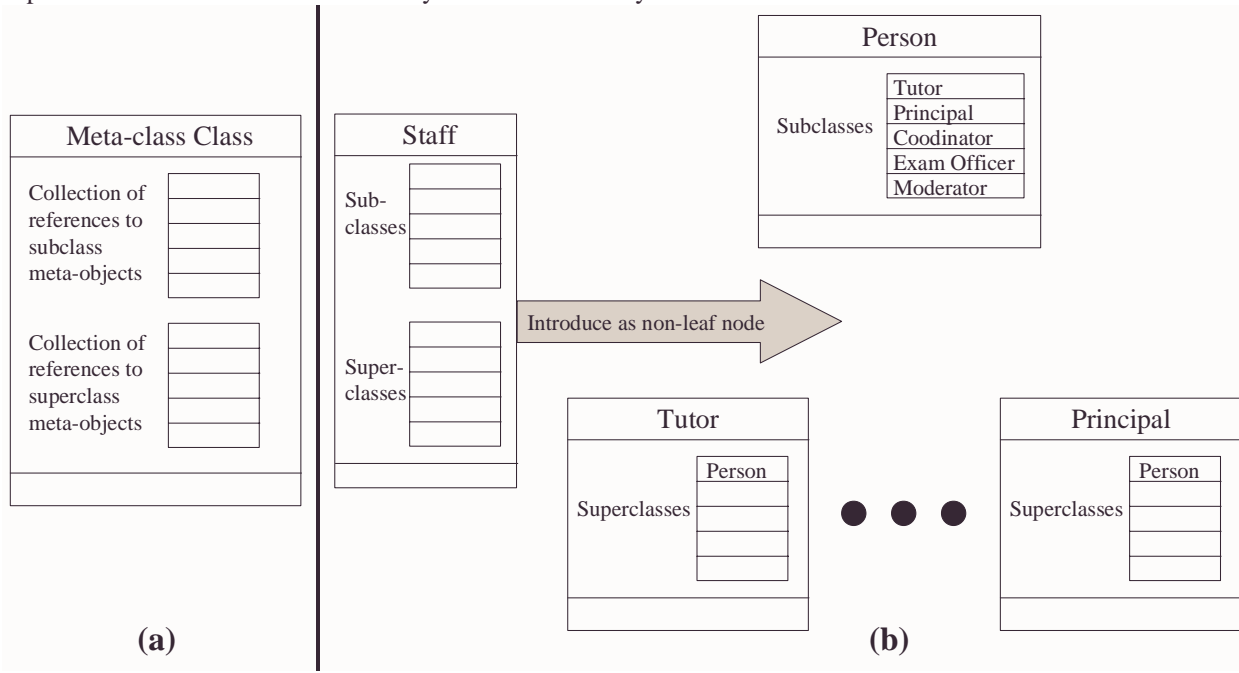


Fig. 10: (a) Meta-class structure in Orion (b) Introduction of a non-leaf class in Orion

Fig. 10(b) shows the meta-objects in such a system prior to the introduction of the non-leaf class *Staff* in the evolution scenario from the case study. All the references to subclass meta-objects will have to be removed from the *Person* meta-object and all the subclass meta-objects will have to be updated to remove the reference to *Person* in their respective collections of superclass references. A reference to the *Person* meta-object will be added to the superclasses collection and references to older subclasses of *Person* will be added to the subclasses collection in the *Staff* meta-object. A reference to the *Staff* meta-object will be added to the subclasses collection (not shown in fig. 6.8) in *Person* and to the superclasses collection in each of its subclasses. In other words such an approach to implementing relationships results in the relationship information being spread across $m+n$ entities where m and n are the cardinality of each relationship edge. The total number of meta-objects affected in Orion will, therefore, be 16 (*Person*, *Tutor*, *Principal*, *Exam_Officer*, *Moderator*, *Staff* and meta-objects for each of the attributes defined in *Staff*). In case of ENCORE and TSE the total number of entities affected will be 21 and 22 respectively (taking into account the updating and modification of relationships among class versions and virtual classes). Note that all the above figures represent an optimised update to each meta-object whereby duplicate updates are carried out in one operation. A similar optimisation is applied in case of SADES as discussed below.

In SADES the links among all persistent entities, and hence the meta-objects forming the schema, are implemented as relationship constructs which are first class objects and encapsulate information about connections among the entities. This results in connection information being separated from the entities localising changes to these

connections. This is in direct contrast with ORION, ENCORE and TSE, which embed connection information within the entities (cf. fig. 10 (a)) hence spreading the relationships across them. In SADES this information is separated and encapsulated in the relationship objects. Note that these relationship objects exist for all types of links i.e. *derives-from/inherited-by* links among classes or individual class versions, *predecessor/successor* links among class versions, *defines/defined-in* links among class versions and attribute definitions, etc. A composition filters mechanism (an AOP technique) [3] is employed to ensure that any messages manipulating links among entities are received by the relationship objects and not by the entities connected by them. Composition filters are ideal candidates for situations requiring message interception and attachment on a per-instance basis [10]. In SADES they are implemented as first class objects hence not requiring any extensions to the basic system architecture. As shown in fig. 11 a dispatch filter intercepts all incoming relationship manipulation messages and delegates them to the relationship objects. Since the relationship information is no longer embedded within the participating entities it can be modified in an independent fashion. It is also possible to introduce new relationships or remove existing ones with localised changes. This results in cost-effective schema modifications. It also improves the extensibility of the system as introduction of new meta-classes only requires introduction of new relationship objects. A detailed description of the structure and semantics of the relationship objects and their effectiveness during schema evolution in SADES can be found in [13].

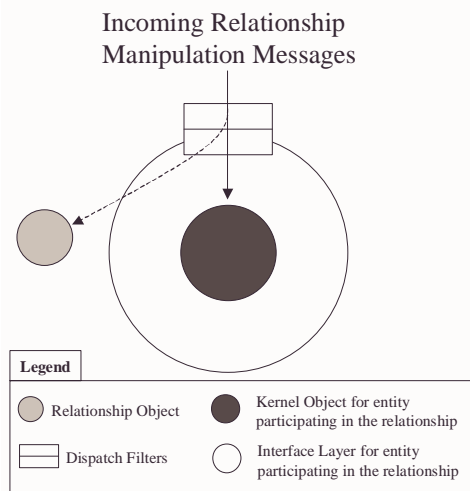


Fig. 11: Implementation of links among entities in SADES using relationship objects and composition filters

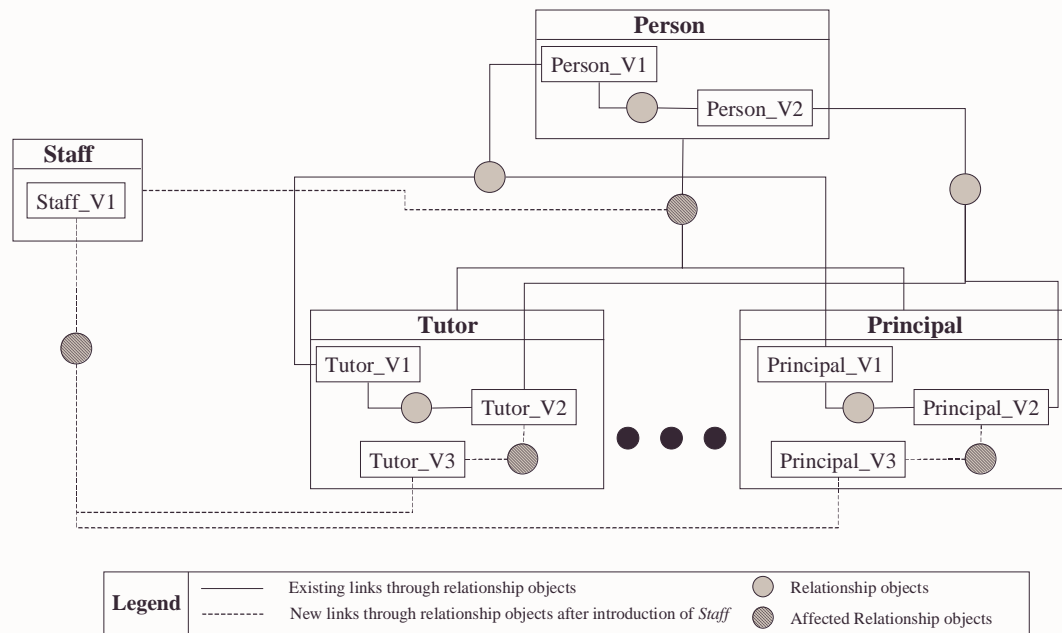


Fig. 12: Introduction of a non-leaf class in SADES

Fig. 12 shows the number of entities affected in SADES upon introduction of the non-leaf class *Staff*. The only entities affected are the relationship objects maintaining the inheritance relationship between *Person* and its subclasses, relevant inheritance relationships at the class version derivation graph level and version derivation relationships between class versions. The total number of entities affected is 8, all of these entities being relationship objects (including one maintaining the relationship between *Staff* and its 9 attributes).

Using the evolution scenario realisations in section 3, table 1 compares the number of entities affected in each of the four systems in response to each of the four general evolution operations outlined in section 2. While at first sight it might appear that the cost-effectiveness of changes in Orion is comparable to that in SADES, this is not the case. Orion employs a much simpler *schema modification* model for evolution with no track of changes. This is in contrast with the richer, hybrid model with track of historical schema changes employed by SADES. Consequently, changes in SADES are better localised despite a richer, more complex evolution model. When table 1 is read in conjunction with figures 2, 5, 7 and 9 one can notice that the cost-effectiveness of the aspect-oriented approach in SADES has a direct correlation with the number of entities participating in a relationship: the larger the number of participating entities, the more effective the aspect-oriented approach is in localising the change. It can also be noted that the efficacy of the aspect-oriented approach bears a relationship with the complexity of the schema. The localisation of the impact is significant in the presence of a complex schema involving a large number of relationships such as the multiple levels of inheritance and version derivation relationships in SADES.

Evolution System Evolution Operation	Orion	ENCORE	TSE	SADES
1. Add/remove/rename attributes in <i>Person</i>	11	25	26	9
2. Introduce <i>Staff</i>	16	21	22	8
3. Drop attributes from <i>Student</i>	4	5	5	3
4. Reposition <i>Student</i>	2	3	3	3

Table 1: Number of affected entities in each of the four systems upon modification of schema relationships

The discussion in section 2 pointed out the need for a *move* primitive in order to move attributes across classes without loss of information i.e. moving attributes common to *Staff* objects from *Person* to a new class *Staff*. As mentioned earlier none of the four systems offers such a primitive. If one attempts to implement such a primitive as a customisation then Orion, ENCORE and TSE will need to update the collections containing references to attribute meta-objects and vice versa. In SADES only the relationship object capturing the *defines/defined-in* relationship between the *Person* meta-object and the nine attribute meta-objects being moved will need to be updated. This implies at least a nine-fold improvement over the other three systems should the customisation be carried out. As discussed in section 4.2 it is also possible to emulate the *move attribute* primitive through customisation of the instance adaptation approach.

It might be perceived that the overhead of delegating messages to the relationship objects in SADES outweighs the benefits achieved by localisation of changes to schema relationships. This is not true as the delegation overhead in SADES is minimal as compared to the large number of schema entities affected in the other three systems upon change. The aspect-oriented approach not only provides cost-effective, customisable schema changes but also offers an extensible system architecture. Consider the scenario when a database system needs to be extended with active features [4]. This will require introduction of a meta-class *Rule* used to instantiate rule classes which will in turn be used to instantiate rule objects. In SADES the introduction of a meta-class *Rule* will only require introduction of a relationship object defining the link between rule classes and other classes in the system. In case of the other three system existing meta-classes will need to be appended with new collections to define such a relationship which will have an impact at the meta-object level. In SADES this inherent support for extensibility has been exploited to non-invasively extend the system with a meta-class *Aspect* which describes the structure of persistent instance adaptation aspects in the system (as discussed in section 4.2).

4.2 Instance Adaptation

This section provides an analysis of instance adaptation in SADES in comparison with instance adaptation in Orion, ENCORE and TSE. The modification of the class *Person* in the evolution scenario is used for the purpose as it involves introduction, removal and renaming of attributes. In order to simplify the description attributes defined by subclasses are not considered in the objects associated with *Person*. This simplification is syntactically and semantically correct as *Person* is not an abstract class and can be instantiated directly.

ORION

ORION employs screening to bring existing instances in line with the schema change. As shown in fig. 13 values of attributes dropped from class *Person* are not physically removed from object O_b , the object created before evolution. Instead they are screened from applications. The rename operation is *information preserving*. The newly introduced *Middle Name or Initial* attribute has a default value³ which is returned when an application attempts to access the value of this attribute in O_b . The instance created after evolution O_a reflects the change in the class structure.

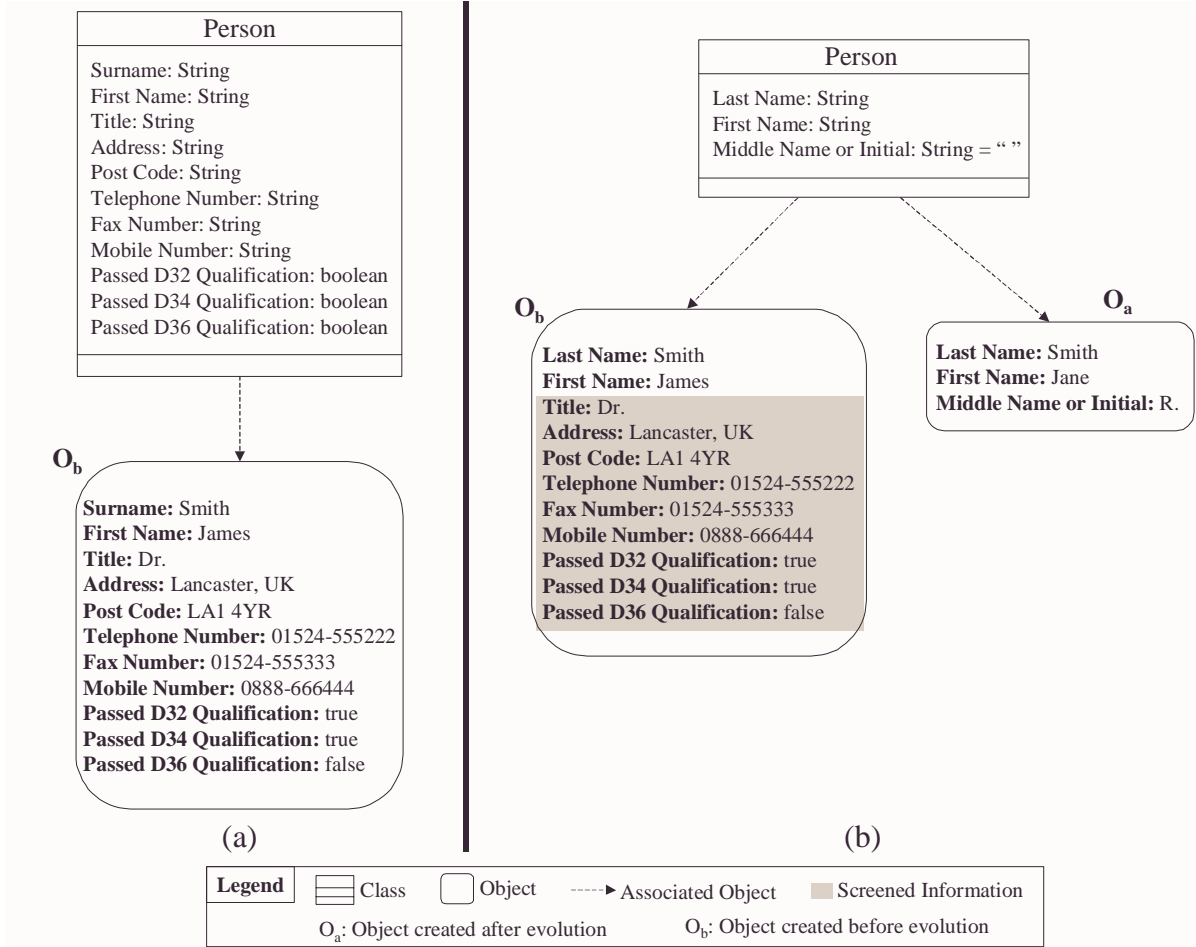


Fig. 13: Instance Adaptation in ORION (a) Before Evolution (b) After Evolution

The above discussion shows that structural changes in ORION are forward compatible. Backward compatibility is possible by using the default values on attributes and screening removed attribute definitions in classes instead of removing them physically. The instance adaptation strategy is fixed and cannot be replaced or customised to suit the needs of the evolution scenario. For example, information contained in attributes moved to *Staff* will be unavailable in *Staff* objects as the instance adaptation strategy does not provide any support for emulating a *move attribute primitive*.

ENCORE

ENCORE employs error handlers to trap incompatibilities between the version set interface (union of the properties and methods defined by all versions of the class) and the interface of a particular class version. These handlers also ensure that objects associated with the class version exhibit the version set interface. If a new class version modifies the version set interface (e.g. if it introduces new properties and methods) handlers for the new properties and methods are introduced into all the former versions of the type. On the other hand, if creation of a new class version does not modify the version set interface (e.g. if the version is introduced because properties and methods have been removed), handlers for the removed properties and methods are added to the newly created version.

³ Although not shown in the fig. all attributes in ORION have a default value.

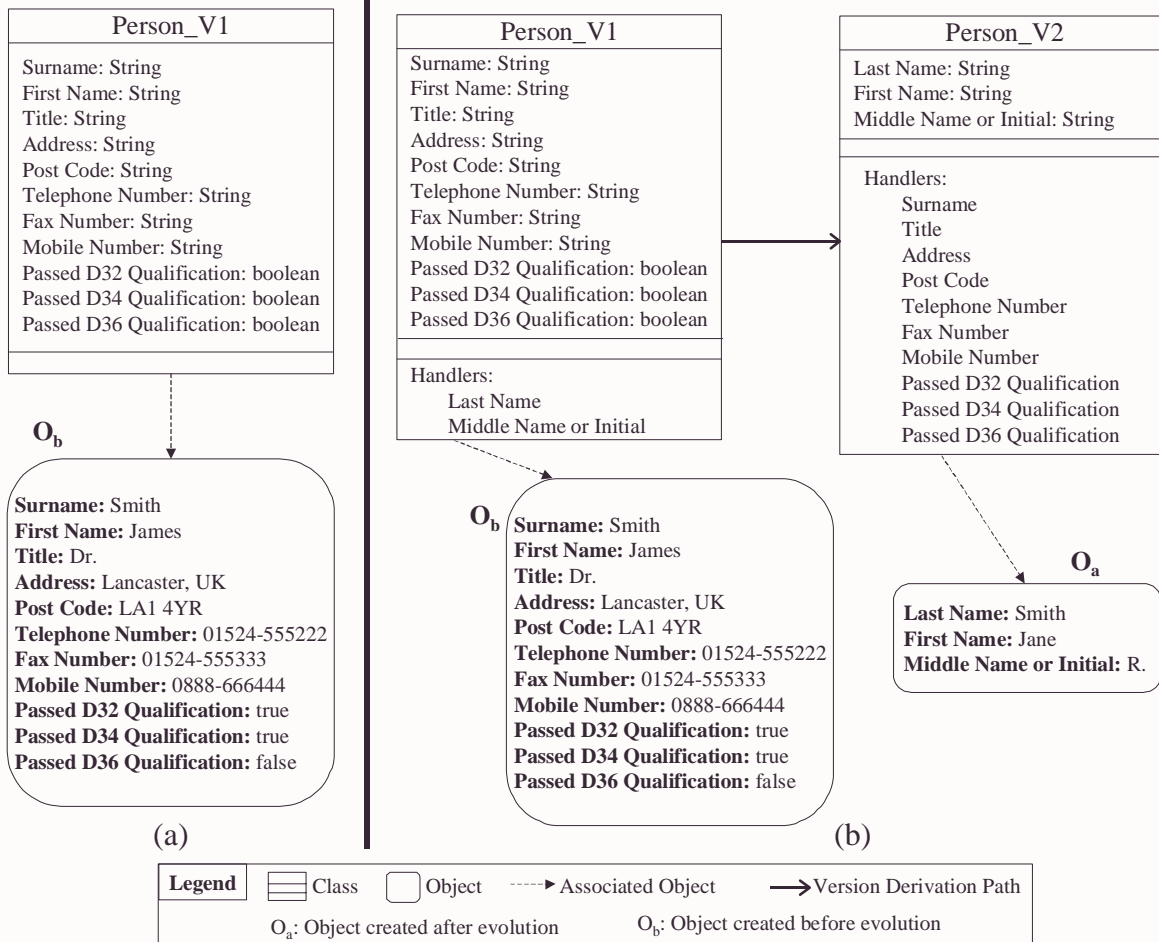


Fig. 14: Instance Adaptation in ENCORE (a) Before Evolution (b) After Evolution

Class version *Person_V2* in fig. 14 modifies the version set interface by introducing a new property *Middle Name or Initial*. A handler for this new property is introduced into *Person_V1* while handlers for the removed properties are added to *Person_V2*. Changes are both forward and backward compatible. However, the instance adaptation strategy is not customisable. As a result realisation of instance adaptation needs specific to an evolution scenario (e.g. emulation of a *move attribute primitive*) are not possible. Furthermore, the handlers are introduced directly into the class versions. Consequently, if several versions of a class exist before an additive change, the handlers for the additive change are copied into all the existing versions. If the behaviour of the handlers needs to be changed all the existing class versions need to be modified.

TSE

TSE employs an instance adaptation approach based on object-slicing complemented by dynamic reclassification, multiple classification and dynamic restructuring of objects. These features can be observed in fig. 15 which shows an object O_b before and after evolution in TSE. As shown in the fig, objects are represented by slices, reflecting the class hierarchy defining their representation. O_b is dynamically restructured to move the attributes now defined by *Person'* into a separate slice. A new slice is created to store values for the attribute *Middle Name or Initial* defined by *Person''*. The slices are arranged in the same hierarchy as the virtual classes and the class *Person* existing prior to evolution and are aggregated by a generic object.

Backward and forward compatibility is maintained. However, the dynamic reclassification and restructuring mechanism can be very costly as a number of the object slices are created due to intermediate changes when realising an evolution scenario and are unlikely to be used extensively. Realisation of scenario specific adaptation (e.g. simulation of a *move attribute primitive*) could be made possible by allowing slices higher in the hierarchy access to data in slices lower in the hierarchy. However, such customisation is not supported.

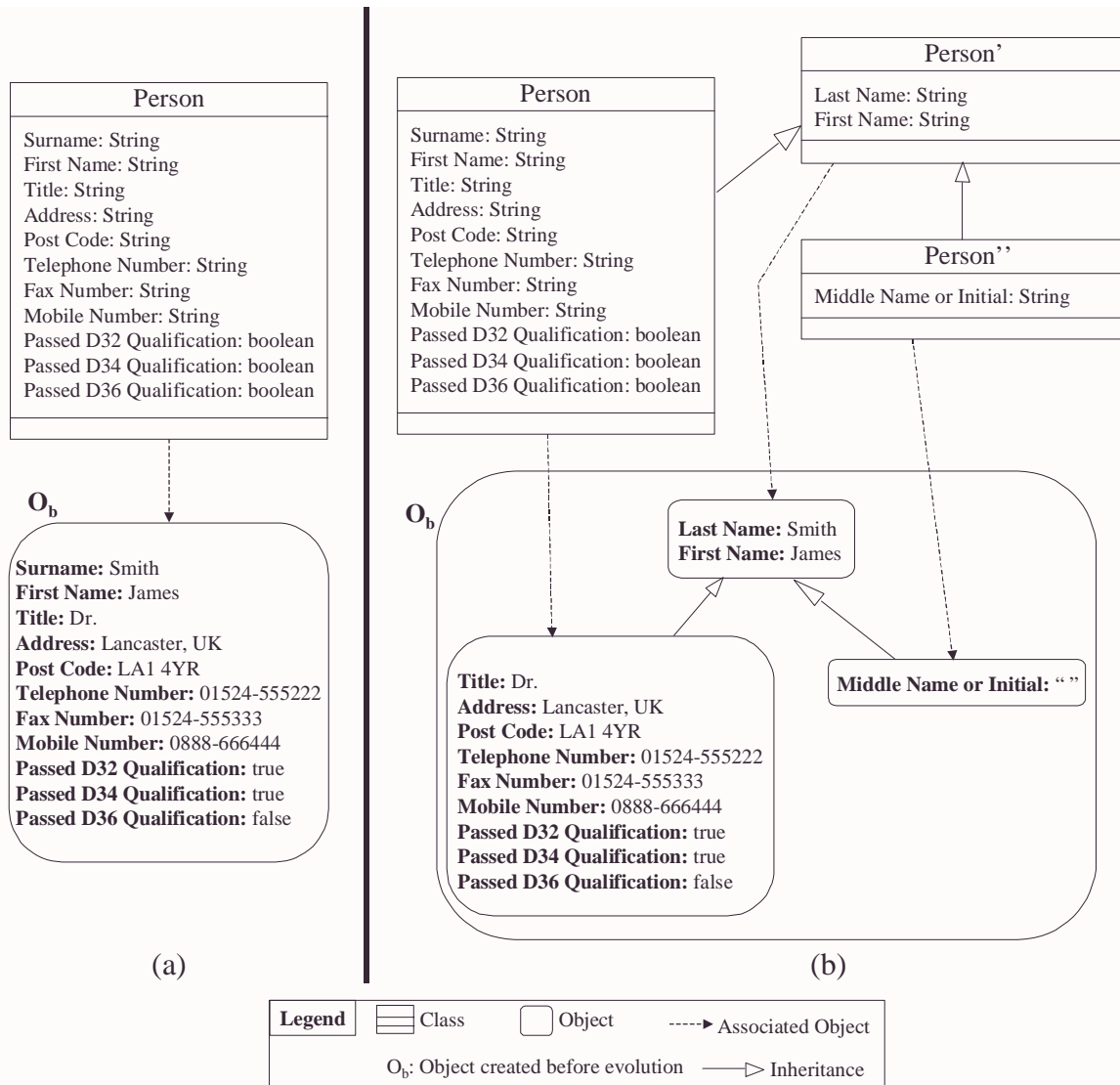


Fig. 15: Instance Adaptation in TSE (a) Before Evolution (b) After Evolution

SADES

As shown in fig. 16 the instance adaptation strategy and adaptation routines in SADES are separated from the classes and their versions using aspects (these are declaratively specified using a simple aspect language modelled on AspectJ [1]). The instance adaptation strategy is exchangeable and can be dynamically woven into the schema manager. The selection of aspects containing the adaptation routines is dependent on the instance adaptation strategy chosen. For example, from fig. 16 aspects containing the handlers will be woven only when the error handlers strategy is woven into the schema manager. If the aspect containing the error-handlers strategy is exchanged with the one containing the update/backdate methods strategy (dynamic instance conversion where an update method specifies what is to happen to an object when converted to a newer class version while a backdate method defines how to convert an object to an older class version) [7], the aspects containing the handlers will be exchanged with those containing the update/backdate methods. Note that the choice of instance adaptation strategies is not limited to error-handlers and update/backdate methods. Other instance adaptation strategies can be employed as shown in fig. 16. A detailed description of aspect-oriented instance adaptation in SADES can be found in [17].

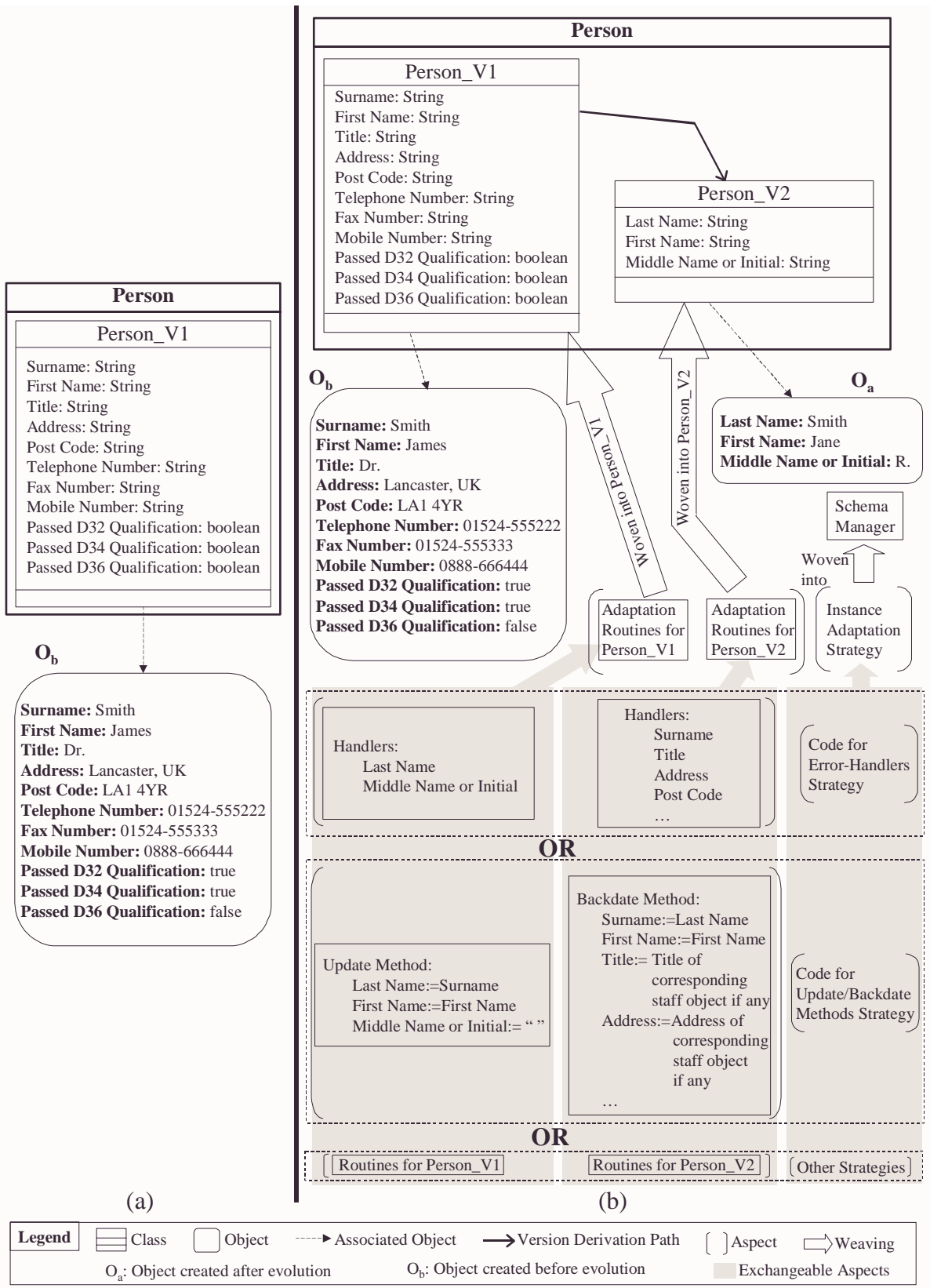


Fig. 16: Instance Adaptation in SADES (a) Before Evolution (b) After Evolution

Separating the instance adaptation code into aspects not only allows changes to the instance adaptation strategy of the system but also provides support for customising the strategy without posing maintenance problems for existing class versions. The changes are local to the aspect and are propagated to the class versions through dynamic weaving. For example, in fig. 16, the backdate method for objects associated with *Person_V2* can test whether the object being associated with *Person_V1* is a *Staff* object. If this is the case the attributes *Title*, *Address*, etc. can take on the values of the attributes in the *Staff* object. This is a customisation of the update/backdate methods strategy [7] and emulates the *move attribute primitive*. Similarly, when using the error handlers strategy, instead of introducing the handlers directly into class versions upon an additive change they are encapsulated in an aspect. As a result customisation of a handler does not require changes to all the existing class versions. Only the aspect needs to be modified. It is also worth noting that, unlike the other three systems, multiple instance adaptation strategies can co-exist in SADES hence providing support for context sensitive adaptation. For example, one group of objects can be adapted using a simulation based approach such as error-handlers while the other through a physical conversion mechanism such as update/backdate methods. Such conversion can also be application dependent.

A comparative analysis of SADES with ENCORE using the evolution scenario has identified that the number of entities (class versions and/or aspects) affected in case of subtractive changes is the same in both cases. This is because ENCORE introduces handlers into the newly derived version only (to account for missing information in associated objects) in case of a subtractive change. Therefore, the number of entities affected upon customisation is 1. In SADES only the aspect containing the handlers is affected. In case of additive changes the improvement over ENCORE is fourfold. If one assumes that the instance adaptation strategy in ENCORE is customisable then the number of entities affected in case of SADES is three times less than that in ENCORE when the instance adaptation approach is customised. A quantitative comparison with instance adaptation in Orion and TSE has not been possible as presently only the error handlers and update/backdate methods strategies have been implemented in SADES. A comparison of SADES with CLOSQL [7], a system implementing the update/backdate methods strategy, identifies a twofold improvement upon both additive and subtractive changes. The improvement is threefold if one assumes that the instance adaptation strategy in CLOSQL is customisable (which is not the case).

The overhead of dynamic modification and weaving of aspects might question the feasibility of the aspect-oriented instance adaptation approach in SADES. Undoubtedly the efficiency of the weaver has an important role to play for the approach to be effective. On-demand weaving (weaving or reweaving only if the aspect has not been woven before or has been modified since it was last woven) and selective weaving (weaving only the modified parts of an aspect) can significantly improve the efficiency of dynamic weaving in a persistent environment [11]. Currently the SADES weaver offers full support for on-demand weaving but only uses a simple selective weaving algorithm. A more sophisticated algorithm will significantly improve the efficiency of the weaving and, hence, the customisations. In any case, any weaving overhead pays off in terms of a highly flexible instance adaptation mechanism which can be customised to the specific needs of an organisation or application.

5. Conclusion and Future Work

This paper has provided a comparative evaluation of two aspect-oriented features of the SADES object database evolution system with three other systems: ORION, ENCORE and TSE. The evaluation is based on a design correction scenario from an organisation where day-to-day activities revolve around the database. The discussion has demonstrated the cost-effectiveness and resilience of the aspect-oriented approach when faced with both anticipated and unanticipated changes in object databases. However, this is not the only contribution of the paper. The paper has also provided a comprehensive analysis of aspect-oriented techniques in a real world scenario. Such case studies are highly beneficial as they provide a realistic evaluation of the advantages of aspect-oriented techniques over conventional approaches.

The comparative study of modification of schema relationships has demonstrated the effectiveness of the aspect-oriented approach in localisation of changes during evolution, customisation and extension of the system. The approach is particularly effective in the presence of a large number of relationships and participating entities. The comparative study of instance adaptation approaches has demonstrated a lack of customisability in existing systems. It has also highlighted the effectiveness of an aspect-oriented approach in localising changes during customisation of specific adaptation routines and the adaptation strategy. Multiple instance adaptation approaches can co-exist hence supporting context sensitive adaptation.

Future extensions of the work include carrying out in-depth case studies comparing other aspect-oriented features of SADES: versioning, change propagation and referential integrity semantics with existing systems. It is also intended that an aspect-oriented behavioural consistency approach will be developed to bring existing applications in line with schema changes (currently this is implemented through binding applications to specific class versions in SADES). A more comprehensive aspect-oriented evolution framework for object databases is currently in

development [6]. This framework aims to support detailed customisations such as customising or completely exchanging the evolution model itself. We intend to carry out case studies to evaluate the effectiveness of this framework in meeting local organisation needs and changes to them during the lifetime of the database.

References

1. Xerox PARC, USA, *AspectJ Home Page*, <http://aspectj.org/>
2. Banerjee, J., et al., *Data Model Issues for Object-Oriented Applications*. ACM Transactions on Office Information Systems, 1987, **5**(1): p. 3-26.
3. Bergmans, L. and Aksit, M., *Composing Crosscutting Concerns using Composition Filters*. Communications of the ACM, 2001, **44**(10).
4. Dittrich, K.R., Gatzju, S., and Geppert, A. *The Active Database Management System Manifesto: A Rulebase of ADBMS Features*. 2nd Workshop on Rules in Databases, 1995: Springer-Verlag, Lecture Notes in Computer Science 985: 3-20
5. Elrad, T., Filman, R., and Bader, A. (eds.), *Theme Section on Aspect-Oriented Programming*. Communications of ACM, 2001, **44**(10).
6. Green, R. and Rashid, A. *An Aspect-Oriented Framework for Schema Evolution in Object-Oriented Databases*. Workshop on Aspects, Components and Patterns for Infrastructure Software (in conjunction with the first international conference on aspect-oriented software development), 2002
7. Monk, S. and Sommerville, I., *Schema Evolution in OODBs Using Class Versioning*. ACM SIGMOD Record, 1993, **22**(3): p. 16-22.
8. Ra, Y.-G. and Rundensteiner, E.A., *A Transparent Schema-Evolution System Based on Object-Oriented View Technology*. IEEE Transactions on Knowledge and Data Engineering, 1997, **9**(4): p. 600-624.
9. Rashid, A. *A Database Evolution Approach for Object-Oriented Databases*. IEEE International Conference on Software Maintenance (ICSM), 2001: IEEE Computer Society Press: 561-564
10. Rashid, A. *A Hybrid Approach to Separation of Concerns: The Story of SADES*. 3rd International Conference on Meta-Level Architectures and Separation of Concerns (Reflection), 2001: Springer-Verlag, Lecture Notes in Computer Science 2192: 231-249
11. Rashid, A., *Weaving Aspects in a Persistent Environment*. ACM SIGPLAN Notices, Feb. 2002.
12. Rashid, A. and Pulvermueller, E. *From Object-Oriented to Aspect-Oriented Databases*. 11th International Conference on Database and Expert Systems Applications (DEXA), 2000: Springer-Verlag, Lecture Notes in Computer Science 1873: 125-134
13. Rashid, A. and Sawyer, P. *Dynamic Relationships in Object Oriented Databases: A Uniform Approach*. 10th International Conference on Database and Expert Systems Applications (DEXA), 1999, Lecture Notes in Computer Science 1677: 26-35
14. Rashid, A. and Sawyer, P. *Evaluation for Evolution: How Well Commercial Systems Do? ECOOP '99 International Workshop on Object-Oriented Databases*, 1999: 13-24
15. Rashid, A. and Sawyer, P., *Object Database Evolution using Separation of Concerns*. ACM SIGMOD Record, 2000, **29**(4): p. 26-33.
16. Rashid, A. and Sawyer, P., *Aspect-Oriented and Database Systems: An Effective Customisation Approach*. IEE Proceedings - Software, 2001, **148**(5): p. 156-164.
17. Rashid, A., Sawyer, P., and Pulvermueller, E. *A Flexible Approach for Instance Adaptation during Class Versioning*. ECOOP 2000 Symposium on Objects and Databases, 2000: Springer-Verlag, Lecture Notes in Computer Science 1944: 101-113
18. Skarra, A.H. and Zdonik, S.B. *The Management of Changing Types in an Object-Oriented Database*. 1st OOPSLA Conference, 1986: 483-495