

DynaSchema: A Library to Support the Relational Data Schema Evolution for the Self-Adaptive Software Domain

Gabriel Nagassaki Campos^a and Frank José Affonso^b

*Department of Statistics, Applied Mathematics and Computation, São Paulo State University – UNESP,
PO Box 178, Rio Claro, São Paulo, 13506-900, Brazil*

Keywords: Self-Adaptive Software, Data Schema Evolution, Reference Architecture.


Abstract: The development of self-adaptive software (SaS) represents a significant challenge, as this type of software enables structural, behavioral, and context changes at runtime. Among the range of SaS, this paper focuses on a specific type of SaS that enables data schema evolution (DSE) at runtime. This type of SaS requires data storage while preserving the integrity between the logical model (i.e., SaS) and the data model (i.e., data schema). Regarding DSE, a solution must encompass not only the migration of the original data model to a new one but also the migration of data from the old schema to the new one without affecting the SaS regarding incompatibility and/or lack of data integrity. Although relevant to the SaS domain, DSE is a research topic that still needs further investigation to develop a comprehensive and robust solution. The objective of this paper is to contribute to this research topic by presenting DynaSchema, a library that enables the evolution of relational data schemas at runtime through a non-intrusive approach. To demonstrate the applicability of the DynaSchema library, a case study was conducted. The findings suggest that the library has the potential to make a significant and efficient contribution to the SaS domain.


1 INTRODUCTION

The role of software systems in modern society has been of essential importance, facilitating the automation of tasks across numerous areas. These include public and private institutions, communication systems, airports, entertainment media, and other application areas. Therefore, it is of the utmost importance that these software systems possess the capacity to address uncertainties that may originate from a multitude of sources, including alterations in their operational environment or changes in the requirements of their users. In this regard, self-adaptive software (SaS) represents a special type of software system, distinguished by its capacity to respond to changes in a dynamic environment. This encompasses the capacity to adapt to evolving user requirements or to autonomously adjust in response to alterations in the execution environment, including instances of degraded quality, overloaded operations, and other forms of adversity (Salehie and Tahvildari, 2009; Weyns, 2019).

From another perspective, reference architectures (RA), a special type of software architecture, have be-

come an important element in the systematic reuse of architectural knowledge. Consequently, a variety of software systems domains, including SaS, have recognized the necessity of encapsulating knowledge (i.e., experiences and best practices) for the purpose of disseminating and reusing this knowledge in the development of systems (Bass et al., 2012). To illustrate, Camargo et al. (2024) designed an RA for the self-adaptive cyber-physical system, named RA4Self-CPS, Affonso et al. (2019) developed an RA for self-adaptive, service-oriented mobile applications, called RA4Self-MobApps, and Affonso and Nakagawa (2013) proposed a RA for SaS, named RA4SaS. Regarding the last architecture, Affonso et al. (2024) presented a second release, in which they highlighted a tool for SaS modeling through a domain-specific language (DSL) named eLanguage. Concerning design, it is notable that, although the aforementioned architectures were originally conceived to serve a specific purpose, they have been developed in a way that allows them to be adapted for use in a variety of different systems. This is achieved through the integration of three key principles: (i) modular organization; (ii) an external adaptation approach; and (iii) the use of computational reflection. The modular organiza-

^a  <https://orcid.org/0000-0001-9737-0734>

^b  <https://orcid.org/0000-0002-5784-6248>

tion enables the reuse of knowledge, or modules, between such architectures. The external adaptation approach provides a means of designing adaptive software independently of the specific adaptation mechanisms involved. Finally, computational reflection has been demonstrated to be a valuable resource for discovering SaS information at runtime.

Among the SaS that can be found in both academia and industry, this paper focuses on a type of SaS that requires data storage while preserving the integrity between the logical model (i.e., SaS) and the data model (i.e., data schema) during execution and adaptation cycles. As argued by Störl et al. (2020), Hillenbrand et al. (2022) and Campos (2024), a solution for data schema evolution must encompass both the schema and data migration in a way that does not affect the software, either in terms of incompatibility or lack of data integrity. As evidenced by the study conducted by Campos (2024), data schema evolution remains a research topic that requires further investigation within the SaS domain, as there is currently no comprehensive and robust solution. Based on this context, the objective of this paper is to present DynaSchema, a library that enables the evolution of relational data schema at runtime. To do so, this library was designed to act as a middleware layer between SaS (i.e., software entities designed by RA4SaS) and the relational databases through a non-intrusive approach. In order to evaluate the applicability of DynaSchema, a case study was conducted using a seller system via the Internet. As a result, the library proposed in this paper offers promising potential for contributing to SaS, providing a feasible alternative for addressing data schema evolution at runtime.

Based on the presented scenario, the library proposed in this paper aims to consolidate itself as a means to facilitate the data schema evolution at runtime. In short, the main contributions of DynaSchema can be summarized as follows: (i) design independence, as it does not interfere with the SaS development. In other words, the software engineer can concentrate on designing software entities without concern for injecting cross-cutting interests into them to ensure the functioning of the DynaSchema library; (ii) modular organization, as it tends to facilitate the expansion of its capacity to accommodate other types of databases (e.g., NoSQL); and (iii) ease of coupling, as it was designed based on a request and notification system. In essence, the adaptation engine (e.g., RA4SaS) must be responsible for forwarding a request for adaptation in the database to the library. Next, the DynaSchema library must then notify the adaptation engine, providing the result of the aforementioned request. Besides contributions related to

the library, this paper also provides a set of requirements in Section 4.1, which may serve as a robust and solid foundation for guiding the development or enhancement of new solutions for data schema evolution for SaS or other software domains. Thus, it is expected to create a favorable scenario for the development of SaS, since the DynaSchema provides an easy means to deal with data schema evolution while enabling the SaS side to evolve and scale.

The paper is organized as follows: Section 2 introduces the essential concepts related to SaS, RA4SaS, data schema evolution, and an overview of related work; Section 3 addresses a running example; Section 4 presents the DynaSchema, a library for the evolution of relational database schema for SaS; Section 5 describes a case study to show the applicability of DynaSchema; Section 6 shows a brief discussion of results; and Section 7 summarizes the conclusions and perspectives for future work.

2 BACKGROUND AND RELATED WORK

This section presents the background and related work that contributed to the development of this paper. First, the concepts of SaS, RA4SaS, and data schema evolution are introduced. Next, related work on the research topic of this paper is addressed.

Self-Adaptive Software. SaS is a special type of software system that is capable of automatically modifying itself in response to changes in its operating environment (Krupitzer et al., 2015). As evidenced by Salehie and Tahvildari (2009), the modification of the internal dynamics of SaS can be achieved through changes in its various artifacts or attributes, which can be executed with minimal human intervention or without any human involvement whatsoever. According to Weyns et al. (2013), changes in the execution environment of SaS may arise due to failures, fluctuations in available resources, shifts in user priorities, and other factors. To overcome such challenges, SaS includes specific features known as “self-properties”, which provide flexibility within the application and help mitigate anticipated variations during its operational phase (Salehie and Tahvildari, 2009).

RA4SaS. This RA was designed to facilitate the development of general-purpose SaS (Affonso and Nakagawa, 2013) based on three main concepts: (i) the MAPE-K (Monitor, Analyze, Plan, Execute over a shared Knowledge) adaptation loop (IBM, 2005), which enables the management of SaS at runtime; (ii) computational reflection (Maes, 1987), which provides the means to modify software entities at run-

time; and (iii) the external adaptation approach (Salehie and Tahvildari, 2009) proposes an adaptation logic organization comprising two layers: (1) **supervisor**, which contains the adaptation logic, and (2) **supervised**, which contains the SaS. Moreover, it is essential to underscore that this architecture operates with a controlled adaptation modality, wherein the developer must specify the adaptation level of each software entity. In its most recent release, RA4SaS enables software engineers to design software entities within a tool designated as DSLModeler4SaS through a DSL namely eLanguage. According to the RA4SaS's features, engineers can use adaptation and persistence annotations to deal with these development concerns (Affonso et al., 2024).

Data Schema Evolution. According to an investigation conducted by Campos (2024), the evolution of data schema is a complex and broad subject that encompasses a multitude of concepts. Given the limitations in terms of space and scope, this section will focus on the principal concepts of data schema and data migration.

As stated in Elmasri and Navathe (2019), a *data schema* represents a formal description of the structure of a database, typically delineated during (or as part of) the software design process. In the view of Roddick (1995), schema modification occurs when a database system enables modifications to the schema definition of a populated database.

According to Störl et al. (2020), the process of *data migration* typically occurs subsequent to the evolution of the underlying schema. In the existing literature, a variety of strategies have been proposed, including eager, lazy, incremental, and predictive approaches (Hillenbrand et al., 2022). In the first strategy, upon the detection of a change in the data model, all entities are migrated to the new model. The second strategy may be the opposite of the first, where data is lazily migrated as it is accessed in the new data model release. The third strategy represents an intermediate between the first and second. It treats changes in the data model as a lazy migration, gradually accumulating migration debt that is addressed by eager migration periods. The fourth strategy aims to monitor historical access to data and its constituent entities, ensuring the timely updating of data deemed "hot" by the system.

As related work, a synthesis of the investigation conducted by Campos (2024) is addressed, which served as the foundation for the design of DynaSchema. It is noteworthy that all studies presented here addressed initiatives related to the evolution of data schema and the migration of data. Next, the main studies of this investigation are presented, categorized

by solution type.

A **Database Management System** (DBMS) represents a solution type that proposes the creation of a DBMS that implements data schema evolution as a native feature. In this direction, Neamtiu et al. (2013) developed a solution that represents a modified version of the SQLite database. In short, this solution enables the introduction of new SQL commands to create new data schema versions. To do so, it exploits the march-forward nature of many database applications, providing on-the-fly updates while ensuring consistency and transparency to database clients. Furthermore, the proposed solution eliminates the need to support old schema versions after the schema update.

Solutions classified as **Tool** address the implementation and use of a tool to facilitate the evolution of the data schema in a software system. JAVADAPTOR is a solution developed by Pukall et al. (2013) that enables the updating of the data schema of a Java application. To do so, the solution combines schema alteration through class replacement by class renaming and caller updates with Java HotSwap, utilizing containers and proxies. These operations are compatible with all major standard Java virtual machines.

Solutions that introduce an intermediate layer between a software system (e.g., SaS) and its database were classified as **Middleware**. In this regard, de Jong et al. (2017) developed a middleware solution that allows multiple versions of an application to store their schemas and share common data in a relational database. To achieve this, the solution employs a mixed-state approach for each schema change set, maintaining a set of synchronized "ghost tables". Hillenbrand and Störl (2023) proposed a manager that enables the management of schema evolution and data migration from a non-relational database. This solution uses heuristics to estimate the impact of schema evolution when the migration situation can be elicited. In contrast, schema migration strategies are used in order to comply with service-level agreements (SLA).

Finally, **Framework** represents a solution type based on a reusable infrastructure of components designed to facilitate the evolution of the data schema in a software system. To illustrate, Beurer-Kellner et al. (2023) has proposed a framework that facilitates data sharing between web services subject to frequent evolution. In short, this framework provides a version-aware interface definition language (IDL) for API (Application Programming Interface) design. This comprises a typed JavaScript-based language for defining migration functions using the IDL definition and a runtime environment for executing migrations.

Despite the significant initiatives mentioned, no solution has yet emerged that adequately supports

SaS development and addresses data schema evolution comprehensively and robustly. The studies presented in this section address the data schema evolution in particular approaches, focusing on the solution in a specific type of target application. However, it is observed that the authors did not consider the minimum essential requirements (see Sections 2 and 4.1) that this type of solution must implement.

3 RUNNING EXAMPLE

This section presents a running example that illustrates the evolution of data schema and data migration related to a software entity undergoing an adaptation process, as shown in Figure 1. In short, the purpose of this example is to facilitate an understanding of how the integrity of software entities can be maintained throughout the adaptation process at runtime. The maintenance of integrity aims to ensure that the goals, domain requirements, database schema, and data of each entity can be accommodated by the changes made during the life cycle of these entities. Moreover, it is also important to note that although this example has been developed with relational databases, the fundamental concept behind it can be applied to non-relational databases as well. Next, a description of this example is addressed.

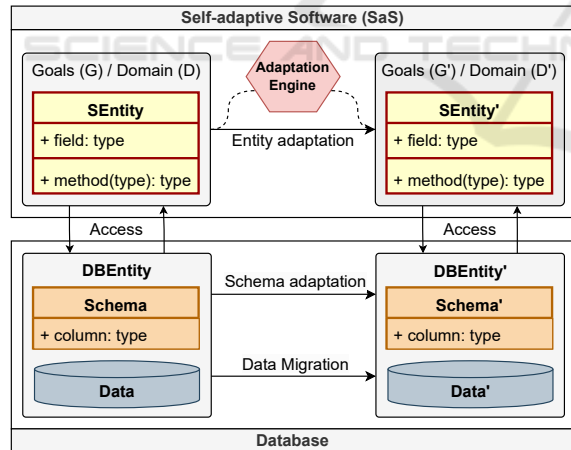


Figure 1: Running example.

In order to illustrate the proposed adaptation scenario, the example was organized into two layers. The top layer represents the software entities (i.e., SEntity and SEntity'), which is a generic term used to refer to SaS. The bottom layer symbolizes database entities (i.e., DBEntity and DBEntity'), which are represented by a table in a relational database and its data. In short, the objective is to design an entity that meets a set of specified goals

(G) and operates within a designated software domain (D). To illustrate this adaptation scenario, the entity designated as SEntity must be adapted, and a new entity named SEntity' must be created.

As illustrated, the Adaptation Engine component intercepts the entity's adaptation, facilitating the process through an external adaptation approach. In essence, the Adaptation Engine changes the adaptable software, referred to as SEntity, to introduce, update, or remove features and/or behaviors in the new entity that will be generated (i.e., SEntity'). This enables the new entity with the capacity to align with the shifting goals (G') or domain (D') of its stakeholders. This may result in a schema adaptation from DBEntity to DBEntity', as well as data migration from Data to Data'. Regarding the adaptation process, three steps can be conducted, namely: (i) software entity adaptation; (ii) data schema adaptation; and (iii) data migration. From an operational viewpoint, the adaptation scenario presented in this example encompasses two distinct scenarios, namely: **(A)** When it is necessary to maintain the two entities in operation. Here, it is necessary to maintain the functionality of both entities (i.e., SEntity and SEntity'). This entails the preservation of the data schemas associated with these entities, which must remain active and accessible to facilitate data sharing. It can thus be stated that only two adaptation steps (i and ii) were executed. **(B)** When there is no longer any reference to the original entity. Here, it is determined that there are no longer any requests for the original entity (i.e., SEntity), with all executions pertaining solely to the new entity (i.e., SEntity'). Consequently, the data belonging to the original entity's data schema can be migrated to the new entity's data schema. Therefore, it can be stated that all three steps (i, ii, and iii) were executed.

4 DynaSchema

This section presents the DynaSchema, a library designed to facilitate the evolution of relational data schema for the SaS domain. To do so, this library implements a flexible data persistence mechanism based on the JPA (Java Persistence API) specification. Thus, this mechanism enables the use of object-relational mapping (ORM) for the development of software entities. In order not to interfere with the SaS development (i.e., software entities designed by RA4SaS (Afonso et al., 2024)), the library was designed through a non-intrusive approach, implemented as a middleware layer between SaS and the underlying database. Regarding the design, the DynaSchema library was

developed based on a set of requirements gathered through a literature mapping conducted by Campos (2024). Section 4.1 provides an overview of such requirements, and Section 4.2 presents the architectural view for the DynaSchema library in a model organized in six modules.

4.1 DynaSchema Requirements

This section presents the requirements that were gathered from the study conducted by Campos (2024) and that served as the fundamental base of the design of the DynaSchema library. Next, a description of each requirement is addressed.

R1. A solution for data schema evolution must be designed to facilitate the transformation of SaS's internal structure while maintaining access to the data previously stored in a relational database. A process for this solution type entails two distinct schemas: (1) an object-oriented schema that reflects the SaS structure, specifically the software entities (Mitranont and Fugkeaw, 2006); and (2) a database schema that reflects the structure of the columns and tables utilized for data storage (Santos et al., 2011). To illustrate the complexity of these operations, two scenarios will be presented, namely: the decomposition of classes and the breaking of relationships. The first requires the creation of a new table for each newly defined subclass, besides the definition of foreign keys to facilitate the relationship between these tables (i.e., columns that store attribute data). The second aims to remove the tables representing the subclasses and insert their columns into the table representing the superclass (Götz and Kühn, 2015).

R2. A solution for data schema evolution should facilitate the migration of the application's data schema to be conducted at runtime. The investigation conducted by Campos (2024) revealed that the schema migration process becomes increasingly costly and may impose some form of downtime on the data alone as the volume of data increases significantly. To overcome these adversities, it is recommended that asynchronous mechanisms must be adopted for the migration of the application's data schema together with notification messages. Therefore, SaS can request a schema migration, and the solution notifies the user when this process begins and ends. Solutions based on the runtime concept facilitate the execution of insert, update, and delete operations by SaS while schema migration is in progress. To do so, such solutions must implement a queue, where the migration is only complete when all queue actions are completed. However, the data query operation requires the solution to inform SaS when the migration process has

ended, preventing the system from recovering partial data (Marks and Sterritt, 2013; Hillenbrand and Störl, 2023).

R3. During the SaS's life cycle, it may be possible that it has to alter the structure of an internal component itself and, next, insert it into the operational environment. Therefore, it can be stated that there is a transient state in which parts of the system may utilize the previous version of the data schema, while the migrated components may operate with the most recent iteration of this schema. In order to meet this requirement, the solution must facilitate SaS's interaction with its data, even in a mixed state that requires interaction with both the old and new versions of the data schema (de Jong et al., 2017). To do so, the solution must incorporate a data synchronization mechanism to ensure that changes made to the old schema version are reflected in the new version and vice versa (Wang et al., 2012).

R4. As previously stated in Section 2, SaS must prompt the alteration of the data schema in accordance with the adaptation executed in the software entities. In some cases, it may be necessary to map data between software entities and their corresponding tables in the database. To illustrate this scenario, a table that contains a numeric column for data categorization will be considered, where each category is represented by a positive integer number. In order to optimize the semantics of this information, the new data schema requires that this column be of the text type, with each category being labeled by a name. Therefore, SaS must establish a mapping between the old numerical category and its equivalent label (Namdeo and Suman, 2021).

R5. As outlined in Section 2, the development of SaS is a challenging task due to the numerous issues that must be addressed with the objective of enabling the software to execute modifications at runtime. Analyzing relational databases available in the literature, it was noted that these databases have different SQL dialects. For instance, an auto-increment column in Microsoft SQL Server database must be defined as `IDENTITY`, and the same feature is defined in the MariaDB database as `AUTO_INCREMENT`. Therefore, it can be stated that a solution capable of handling different types of relational databases would be advantageous (Namdeo and Suman, 2021). To do so, a solution must be designed based on a flexible design capable of allowing new SQL dialects to be inserted without significant changes in its source code (Beurer-Kellner et al., 2023).

R6. During the evolution of a software entity (i.e., SaS), the database schema may also undergo changes, as may the data associated with it. It is thus possible to

have both old and new entities running concurrently, each with its own database schema. The functionality that emerges from this scenario is the ability of a SaS to recognize and define the trigger for migrating data from the old schema to the new one. Therefore, the solution for data schema evolution must also provide notification to the SaS regarding the status of the schema evolution and data migration. This is necessary for the SaS to recognize that the old entity has evolved into a new one and that there are no users connected to it, allowing for the definitive schema and data migration to be performed. Otherwise, an incomplete schema migration could cause the SaS to break or execute with inconsistent data (Wang et al., 2012).

4.2 DynaSchema Architecture

Figure 2 illustrates the DynaSchema architectural design in accordance with the specified requirements (R1 to R6) outlined in Section 4.1. As can be observed, the DynaSchema library (dotted line) was designed as a data persistence mechanism, acting as a middleware layer between SaS and the database. This design option enables the library to behave as a non-intrusive solution concerning other layers, specifically the database and the SaS layer. Moreover, the library was structured into a modular format to provide a viable option for stakeholders seeking to design SaS that requires data schema evolution at runtime. From an operational viewpoint, the **Middleware layer** receives a request from the adaptation process (i.e., **SaS layer**) and performs the data schema evolution based on a well-defined sequence process. The **SaS layer** must be notified in relation to the changes made in the **Database layer** so that the software entities can handle their schema properly. Next, a description of each module is addressed.

State Manager. The objective of this module is to facilitate the storage of metadata that will be utilized by other modules within the DynaSchema library for the purpose of coordinating their respective functionalities [R3]. To illustrate, this metadata comprises data regarding the current data schema, the status of ongoing schema migrations, the status of ongoing data migrations, a list of schema and data migration histories, information about database connections, and a list of flags for identifying the status of each schema and data migration. The aforementioned flags facilitate the identification of whether the activities were correctly executed or require rollback. To enable such operations, this module provides a local data repository (e.g., database, XML (eXtensible Markup Language) or JSON (JavaScript Object Notation) text files) to store the aforementioned metadata of each ap-

plication during the data schema evolution process so that other modules can manage them without conflicting access.

Persistence Manager. The principal objective of this module is to address the issue of data persistence in relational databases for each application (i.e., SaS) [R1]. To perform the ORM, this module must have access to the metadata contained in the **Schema Manager** module. This is necessary for it to be able to determine the appropriate mapping type to be conducted, namely: (1) the current data schema or (2) the candidate data schema (mixed state). To facilitate such operations, this module implements reflection-based mechanisms for discovering and generating information at runtime, which is then used for persisting objects in the database and vice versa. Moreover, the mapping strategy incorporated into this module represents a notable distinction from existing conventional ORM solutions documented in the literature. In such solutions, the mapping between classes and tables is typically configured at the outset of the software development phase. On the other hand, this module enables runtime mapping following the current data schema of the **Schema Manager** module.

Schema Manager. The objective of this module is to ensure the effective management of data schema versions (i.e., current and candidate) utilized by an application (i.e., SaS) [R3] throughout the migration process. To this end, the module implements a strategy capable of handling both schemas simultaneously, preventing an incorrect mapping between classes and tables. In essence, the modification actions of the current data schema must be performed independently in a candidate schema, thus preventing any disturbance to the application that owns the current data schema. Furthermore, this module must also be responsible for storing a list of actions for modifying data schemas, enabling the user to request new modifications throughout the application's execution cycle. Once the aforementioned actions have been successfully applied, they can be utilized to facilitate future data schema migration requests, as the sequence of steps has already been defined.

Migration Manager. The principal objective of this module is to facilitate the management of the database schema migration process [R1]. It is important to note that this process encompasses alterations to both the application (object-oriented) and the database schema (relational). To this end, this module coordinates the migration process following three steps: (i) the data schema must be migrated, with the **Schema Migrator** module assuming responsibility for this task; (ii) the data must be migrated [R4], with the **Data Migrator** module assuming responsibility for this task; and

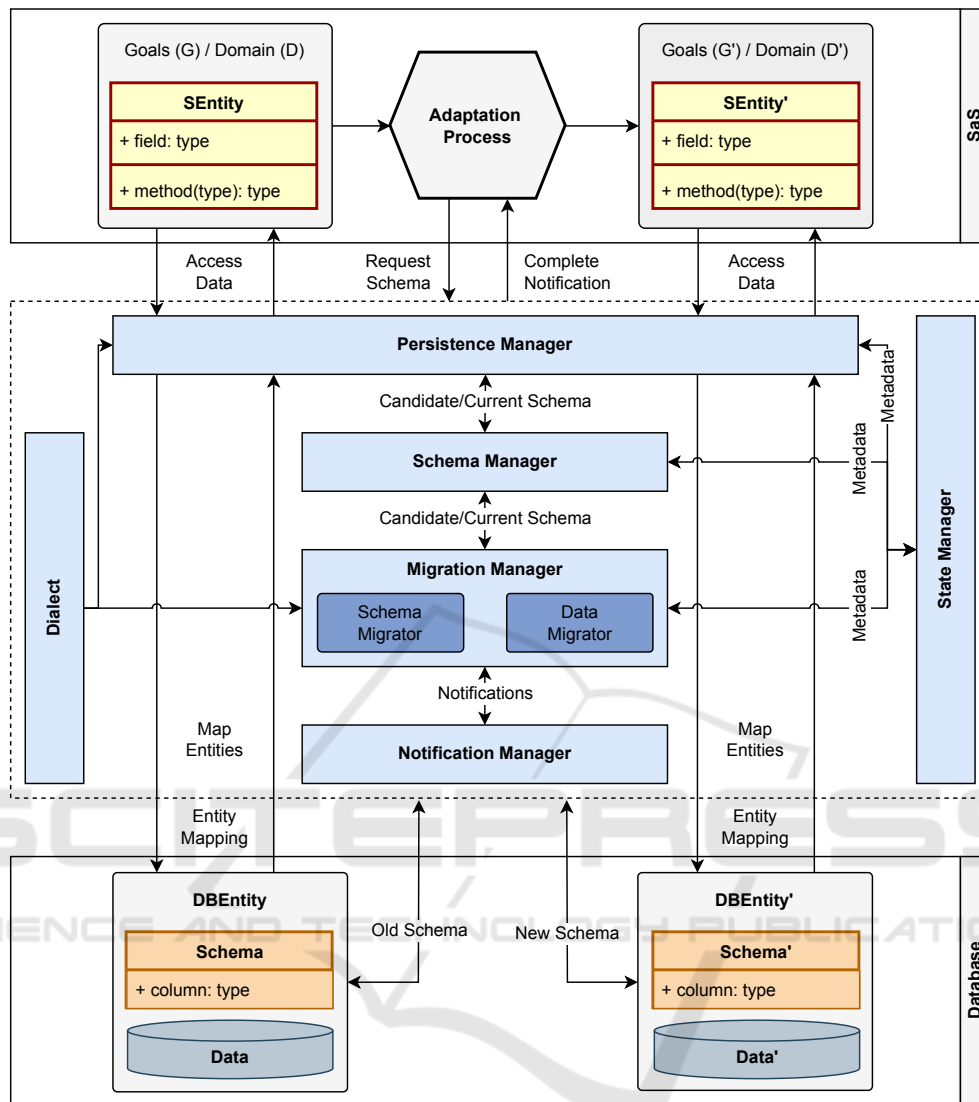


Figure 2: DynaSchema architecture.

(iii) the context must be changed to the current data schema [R6], which requires coordination between the modules **Schema Manager** to change the current schema and **State Manager** to save the changes. Furthermore, the **Migration Manager** module uses the **State Manager** module to store a list of database schema modification actions, thereby enabling the user to request additional modifications. Upon the initiation of a data migration request, the actions accumulated in the **Schema Manager** module can be immediately applied via **Migration Manager** module. In addition, the **State Manager** module is employed to store metadata regarding the ongoing migration process, which is analyzed in three key activities: (1) the completion status of the schema migration, including the identification of any errors, (2) the

completion status of the data migration, including the identification of any errors, and (3) the completion status of both the schema and data migrations, enabling the context to be switched to the most current schema version and data.

Notification Manager. This module is responsible for notifying the SaS of the actions being performed by the DynaSchema library [R2]. An increase in the quantity of data stored in the database may result in delays in the data schema migration process. For this reason, the migration must be performed asynchronously to prevent the execution of SaS from being compromised and/or interrupted. To overcome this adversity, SaS must be informed of the start and end of this task to prevent the migration process from becoming faulty. Thus, it is of essential importance

the capacity to ascertain when a migration process is complete, as partial information may otherwise be recovered if the migration process is still in progress. Furthermore, SaS can also be informed about the process of removing the old version of the data schema, enabling a self-adaptation cycle to be completed.

Dialect. This module is responsible for defining the SQL statements executed by the DynaSchema library [R5]. The software engineer can choose a relational DBMS that best aligns with the requirements of the application domain in question (i.e., SaS). It is important to note that the syntax of an SQL statement can vary significantly between different DBMSs. To address this issue, this module must implement support for SQL dialects to facilitate compatibility with the various DBMSs. Additionally, the software engineer must configure SaS in conjunction with the DynaSchema library to operate with a specific dialect that aligns with their requirements. This configuration must be performed within the metadata stored by the **State Manager** module.

5 CASE STUDY

This section presents a case study conducted to evaluate the applicability, strengths, and weaknesses of the DynaSchema library proposed in this paper. The subject application for this empirical analysis is a web system that manages the sales of a book distributor, referred to from this point on as **SallesSys**. Next, a description of this system and the empirical strategies adopted for conducting this case study are presented.

Subject Application. The **SallesSys** system was developed with the objective of streamlining the process of selling books. To do so, this system has an online catalog that enables customers to access a comprehensive catalog of available titles. With regard to operational procedures, the purchase of these books is permitted only to customers duly registered in **SallesSys**, which can be either individuals or legal entities. To make a purchase, these customers must contact one of the system's salespersons to place an order. In the initial purchase, the salesperson must first register the customer in the system based on the following information: personal data, contact information, and delivery and billing addresses. Next, the salesperson must create a new order comprising the books requested by the customer. After the service, the salesperson must verify the order information with the customer, complete the incorporation of this sales order into **SallesSys**, and transfer the customer to the finance department for the payment of the order.

As illustrated in Figure 3, the development of the

SallesSys system can be summarized in two distinct phases: (1) architectural design; and (2) development activity. Regarding architectural design (**Phase 1**), **SallesSys** was organized in three layers: presentation, application, and database. For reasons of space and scope, this section will focus on the design, development, and adaptation of the software entities (i.e., SaS) within the application layer. At this phase, the software engineer concentrates their efforts on the architectural design of the system, which is based on design patterns, architectural patterns, and architectural decisions. As a result, a set of templates must be developed for automatic code generation, as evidenced by the presence of the `.ftl` files in each layer of the application layer.

Concerning the development (**Phase 2**), the software engineer can utilize the `DSLModeler4SaS` tool to facilitate the development of software entities for the **SallesSys** system. Thus, the software engineer can design software entities and utilize the automated process of RA4SaS to generate source code in a target programming language (i.e., Java). As may be observed in the **SallesSys** project, the engineer creates a `“.entity”` file within the `sallessys.entity` package for each software entity of the **SallesSys** system. Following this phase, the templates developed in **Phase 1** for the code generation can be imported to Templates section. Moreover, it is worth mentioning that the configuration of these templates in relation to the software entities is an essential task for generating the source code. To do so, the `DSLModeler4SaS` tool provides a template configuration wizard that facilitates the specification of the desired configuration in relation to the software entities. Due to the limitations of space and scope, the details of the configuration wizard will not be reported in this paper. However, a more comprehensive understanding of this wizard can be seen in the study by Affonso et al. (2024).

Empirical Research Strategy. To demonstrate the functionalities of the DynaSchema library, it is assumed that the **SallesSys** system was properly implemented and deployed in the execution environment, as described in Section 3. As illustrated in Figure 3, the `Contact` and `Person` entities (see `.entity` files) have ORM annotations for mapping of the software entities to database, besides the adaptation annotations (e.g., `@ClassAnnotation` – Line 1). Based on the RA4SaS persistence module¹, the following annotations are used: `@Entity` (Line 3), `@Id` (Line 5), and `@Relationship` (Lines 8 and 10). Moreover, both entities have been annotated with the adaptation

¹According to Affonso et al. (2024), RA4SaS has a persistence annotation module based on the JPA, which will not be presented in this paper because of space limitations.

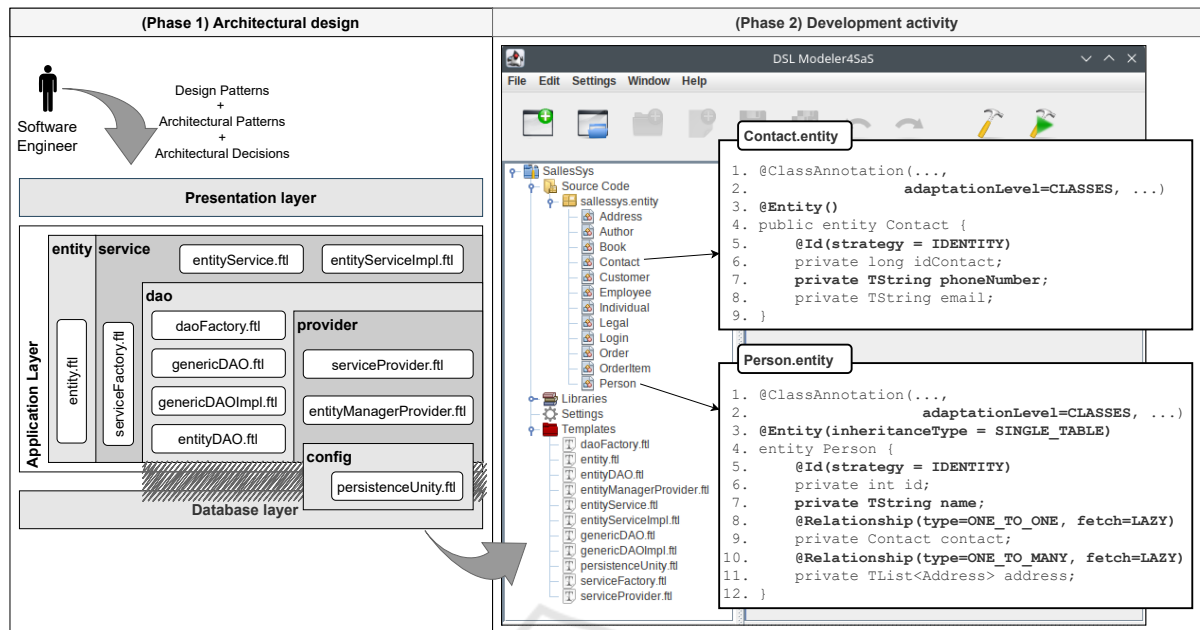


Figure 3: Architectural design and entity development.

level **CLASSES** (RA4SaS adaptation module), indicating that they can undergo structural and behavioral modifications.

As outlined in Section 4.2, the adaptation of a software entity is controlled by an adaptation process (i.e., RA4SaS), which triggers events to the DynaSchema library containing instructions for modifying the application's data schema. As highlighted in bold in the code listings of the *Contact* and *Person* entities (Figure 3 – Line 7), the *phoneNumber* and *name* attributes will be used to characterize the adaptation of a software entity (i.e., SaS) and exemplify how an entity of the data schema, associated with these entities, can be modified to accommodate the new adaptation concerns. In short, such modifications can be summarized in two scenarios: **(A) The addition of attributes:** the *phoneNumber* attribute of the *Contact* entity must be removed and two new attributes (i.e., *homePhone* and *businessPhone*) will be created; and **(B) The breaking of attributes:** the *lastName* attribute will be created in the *Person* entity, allowing it to handle both *name* and *lastName* attributes simultaneously. To illustrate the behavior of the DynaSchema library, it is sufficient to consider the first scenario, as it encompasses the operations of both scenarios and is the most complex in terms of the number of operations. Figure 4 illustrates the sequence of steps and the source code generated by DynaSchema.

The SaS is initiated in its execution environment (**Step 1**), and an adaptation activity is triggered fol-

lowing the interests of the application (i.e., SaS). Following the previous step, the adaptation process (i.e., RA4SaS) must coordinate the calls to the data schema evolution activities provided by the DynaSchema library. To do so, the SaS adaptation process must register itself with the library (Line 1), becoming aware of the notifications triggered by the asynchronous actions taken by the DynaSchema library (**Step 2**).

Steps 3 to 5 represent the changes to the entity, as one attribute is being removed and two attributes are being added. In Line 3, the *phone* attribute is removed from the *Contact* entity, thus allowing for the insertion of the *homePhone* and *businessPhone* attributes (Lines 5 to 9 and Lines 11 to 15). The following code snippets illustrate the declaration of the aforementioned attributes, their designation as either “name” and “type”, the definition of their names as database columns, and their assignment to the *Contact* software entity.

In **Step 6**, the mapping of attributes is executed so that the migration process can be achieved. The *phone* attribute is mapped to *homePhone* (Lines 17 to 20) to facilitate the transfer of existing data to the new entity within the database. All structural changes to the *Contact* entity and its attributes mapping are committed to the candidate data schema (Line 21).

Upon completion of the schema migration (**Step 7**), the data can be migrated from the old schema to the new one (**Step 8**) so that the library can remove the old schema and designate the new one as the current schema for the software entity (**Step 9**).

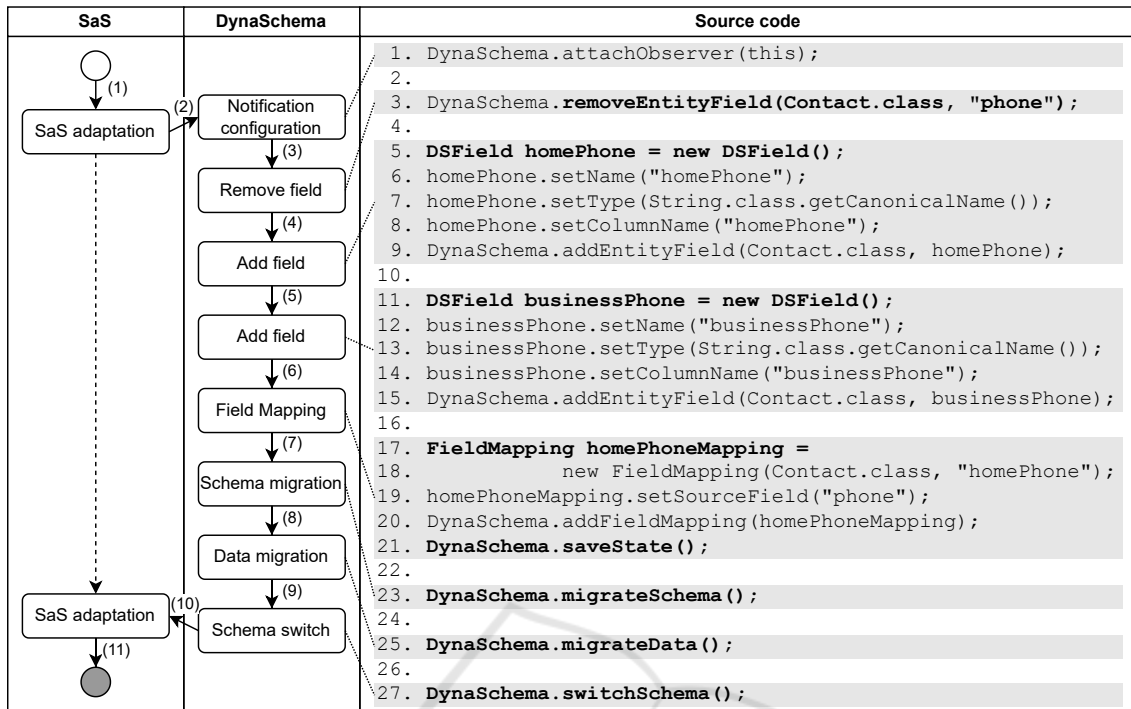


Figure 4: Sequence of steps for data schema evolution.

Upon switching to the new schema, DynaSchema must notify SaS that the activity has been completed (Step 10) so that the software entity (i.e., SaS) can utilize the new data schema (Step 11).

6 DISCUSSION OF RESULTS

This section summarizes the main findings and discusses the relevance of the DynaSchema library to the SaS and Software Engineering communities. The main findings and results are listed as follows.

Independent and Reusable Design. DynaSchema is a middleware-type solution that is capable of handling data schema evolution for SaS. To do so, the library operates as an intermediary between SaS and its database, offering the following functionalities: (i) migration of the relational data schema; (ii) data persistence in the database; and (iii) data migration between schemas. Moreover, its modular organization allows the library to be an extensible solution, capable of supporting other types of databases, and reusable, serving as a reference for the design of new solutions.

Support for Different Data Models. The process of migrating data schemas within the DynaSchema library can be classified into two primary categories: object-oriented and database schemas. The first refers

to the structural configuration of the software entities (i.e., SaS), which can be classified as classes of the object-oriented paradigm within the context of RA4SaS. The second is related to the database schema, which represents the structure of the tables (in the case of relational databases) for the storage of SaS data. In this sense, it can be stated that the DynaSchema library may operate with two distinct data models: (i) the database model, which is associated with the operations to evolve the database schema; and (ii) the object-relational model, which is linked to the ORM approach implemented by the DynaSchema library to address data persistence.

Segmentation of Operations. The data schema evolution process was structured into discrete stages in the DynaSchema library, separately delineating the migration of the database schema and the data contained in such schema. This segmentation enables the library, in conjunction with SaS, to identify the optimal temporal window for the data migration process, thereby avoiding potential issues associated with system interruption or data inconsistency.

Schema Versioning. The DynaSchema library was developed with the objective of providing a solution that could handle the versioning of the SaS data schema over time to avoid any disruption to the SaS execution process. This feature incorporates internal operational mechanisms that safeguard against failures in the SaS data schema evolution process,

thereby preventing adverse impacts on application performance and minimizing instances of downtime. Furthermore, this feature offers a viable method for monitoring SaS during the initiation of a schema evolution operation. The stakeholders of this SaS can trace the evolution process, thereby facilitating any manual reversion process.

Learning Curve. The library proposed in this paper was designed to facilitate the preservation of developers in their native development environment, specifically in the SaS domain. To this end, the cognitive effort required to learn to use DynaSchema is reduced by the presence of a facilitator (i.e., middleware layer) situated between SaS and the database. Therefore, they can focus their attention on the development of the software entities without concern for the injection of source code to address data schema evolution issues.

As detailed in this paper (see Section 4), while the library design incorporated aspects such as design independence and reusability, two considerations warrant particular attention. Firstly, despite adherence to the proposed requirements (see Section 4.1), the library proposed in this paper cannot be considered a comprehensive solution that can address all the needs required by a real-world SaS. Secondly, although DynaSchema was evaluated in a development and execution context associated with RA4SaS (see Section 5), the gathered evidence suggests that this library can be easily adapted for other software domains.

7 CONCLUSIONS

This paper presented DynaSchema, a library that enables dealing with data schema evolution at runtime through a non-intrusive approach. This type of approach enables the integration of DynaSchema with SaS as an intermediary layer between the database and SaS. As presented in Section 4.2, the proposed library addresses all the requirements reported in Section 4.1, which were established based on the study conducted in Campos (2024). The aforementioned requirements provide a robust and solid foundation for guiding the development or enhancement of new solutions addressing data schema evolution in the SaS domain or other software domains. Based on the presented context, the principal contributions of this paper are as follows:

- for the SaS area, by providing a library that facilitates the development of SaS that requires data schema evolution at runtime;
- for the software engineering area, by presenting a set of requirements that served as a solid base for

DynaSchema design, and that may be utilized as a reference for the development of new solutions or enhancement of existing solutions; and

- for different development areas, by providing a solution based on modular organization-based and reusable design. This feature enables the potential extension of DynaSchema to cover other database types (e.g. NoSQL databases).

Regarding future work on DynaSchema, at least three activities are intended: (i) conduction of more case studies or proof of concepts intending to completely evaluate the proposed library, including different adaptation scenarios and databases; (ii) instantiation of DynaSchema for other programming languages to evaluate its structures and respective elements, behavior, and relationships between them when a new homogeneous or heterogeneous computational environment is utilized; (iii) use of this library in the industry to evaluate its behavior when it is applied in a larger real environment of development and execution. Therefore, based on the evidence presented in this paper, a favorable research scenario can be delineated, as it is expected that the proposed library may prove to be an effective contribution to both the software engineering and SaS communities.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), under Grant 88887.670715/2022-00; and in part by the São Paulo Research Foundation (FAPESP), Brazil, under Grant 2017/01703-6 and Grant 2019/21510-3.

REFERENCES

- Affonso, F. J., Nagassaki Campos, G., and Guiguer Menaldo, G. (2024). A reference architecture based on reflection for self-adaptive software: A second release. *IEEE Access*, 12:97476–97499.
- Affonso, F. J. and Nakagawa, E. Y. (2013). A reference architecture based on reflection for self-adaptive software. In *The 7th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS' 13)*, pages 129–138.
- Affonso, F. J., Passini, W. F., and Nakagawa, E. Y. (2019). A reference architecture to support the development of mobile applications based on self-adaptive services. *Pervasive and Mobile Computing*, 53:33 – 48.
- Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition. Published:05 October 2012.

- Beurer-Kellner, L., von Pilgrim, J., Tsigkanos, C., and Kehrer, T. (2023). A transformational approach to managing data model evolution of web services. *IEEE Transactions on Services Computing*, 16(1):65–79.
- Camargo, M. P. d. O., Pereira, G. d. S., Almeida, D., Bento, L. A., Dorante, W. F., and Affonso, F. J. (2024). Ra4self-cps: A reference architecture for self-adaptive cyber-physical systems. *IEEE Latin America Transactions*, 22(2):113–125.
- Campos, G. N. (2024). Dynaschema: uma biblioteca para evolução de banco de dados relacional para o domínio de software autoadaptativo. Master thesis (in Portuguese), São Paulo State University (Unesp), Institute of Geosciences and Exact Sciences (IGCE), Rio Claro. Unesp's Graduate Program in Computer Science (PPGCC), Available in <https://hdl.handle.net/11449/255063>, Accessed on: March 21, 2025.
- de Jong, M., van Deursen, A., and Cleve, A. (2017). Zero-downtime sql database schema evolution for continuous deployment. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP '17*, page 143–152, Buenos Aires, Argentina. IEEE Press.
- Elmasri, R. and Navathe, S. B. (2019). *Sistemas de banco de dados*. Pearson Education do Brasil, 7 edition.
- Götz, S. and Kühn, T. (2015). Models@run.time for object-relational mapping supporting schema evolution. In *CEUR Workshop Proceedings*, volume 1474, page 41 – 50.
- Hillenbrand, A., Störl, U., Nabyev, S., and Klettke, M. (2022). Self-adapting data migration in the context of schema evolution in nosql databases. *Distributed and Parallel Databases*, 40(1):5–25.
- Hillenbrand, A. and Störl, U. (2023). Managing schema migration in nosql databases: Advisor heuristics vs. self-adaptive schema migration strategies. *Communications in Computer and Information Science*, 1708 CCIS:230 – 253.
- IBM (2005). An architectural blueprint for autonomic computing. [On-line], *World Wide Web*. Available in <https://drive.google.com/file/d/1ZY5wMBsugcCoeMCn2GxrX1ZNAr7dUHHT/view?usp=sharing>, Accessed on: March 21, 2025.
- Krupitzer, C., Roth, F. M., VanSyckel, S., Schiele, G., and Becker, C. (2015). A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206.
- Maes, P. (1987). Concepts and experiments in computational reflection. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, OOPSLA '87*, page 147–155, New York, NY, USA. Association for Computing Machinery.
- Marks, R. M. and Sterritt, R. (2013). A metadata driven approach to performing complex heterogeneous database schema migrations. *Innovations in Systems and Software Engineering*, 9(3):179 – 190.
- Mitranont, J. L. and Fugkeaw, S. (2006). Direct access versioning for multidimensional database schema creation. In *The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)*, pages 17–17.
- Namdeo, B. and Suman, U. (2021). A model for relational to nosql database migration: Snapshot-live stream db migration model. In *The 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, volume 1, pages 199–204.
- Neamtii, I., Bardin, J., Uddin, M. R., Lin, D.-Y., and Bhattacharya, P. (2013). Improving cloud availability with on-the-fly schema updates. In *Proceedings of the 19th International Conference on Management of Data, COMAD '13*, page 24–34, Mumbai, Maharashtra, IND. Computer Society of India.
- Pukall, M., Kästner, C., Cazzola, W., Götz, S., Grebhahn, A., Schröter, R., and Saake, G. (2013). Javadaptor - flexible runtime updates of java applications. *Software - Practice and Experience*, 43(2):153 – 185.
- Roddick, J. F. (1995). A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393.
- Salehie, M. and Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2).
- Santos, R. J., Bernardino, J., and Vieira, M. (2011). 24/7 real-time data warehousing: A tool for continuous actionable knowledge. In *The 35th Annual Computer Software and Applications Conference*, pages 279–288.
- Störl, U., Klettke, M., and Scherzinger, S. (2020). Nosql schema evolution and data migration: State-of-the-art and opportunities. *Advances in Database Technology - EDBT*, 2020-March:655–658.
- Wang, Q., Du, Z., and Liu, N. (2012). Design and realization of database online migration. In *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*, pages 1195–1198.
- Weyns, D. (2019). Software engineering of self-adaptive systems. In Cha, S., Taylor, R. N., and Kang, K., editors, *Handbook of Software Engineering*, pages 399–443, Cham. Springer International Publishing.
- Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., and Göschka, K. M. (2013). On patterns for decentralized control in self-adaptive systems. In de Lemos, R., Giese, H., Müller, H. A., and Shaw, M., editors, *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, pages 76–107, Berlin, Heidelberg. Springer Berlin Heidelberg.