Self-adaptive Software: Development Approach and Automatic Process for Adaptation at Runtime

Frank José Affonso [1]
Elisa Yumi Nakagawa [2]

Abstract: O desenvolvimento de um Software autoadaptativo (Sa) é uma tarefa complexa, pois este tipo de software lida constantemente com mudanças estruturais e/ou comportamentais em tempo de execução para que as necessidades de seus usuários ou do ambiente de execução sejam atendidas. Embora existam importantes iniciativas na área de Engenharia de Software para Sa (ES4Sa), incluindo abordagens, processos, métodos, e técnicas para o desenvolvimento de Sa que podem ser encontrados na literatura, as abordagens com suporte automatizado necessitam ser melhor exploradas, pois representam uma alternativa factível para maximizar a velocidade de implementação de tais sistemas e, ao mesmo tempo, para minimizar o envolvimento dos desenvolvedores. Embasado nesse cenário, o presente artigo apresenta uma abordagem com suporte automatizado para o desenvolvimento de Sa. Tal abordagem atua sobre uma modalidade de adaptação controlada, ou seja, os engenheiros de software definem os níveis de adaptação suportados pelo Sa (ou seja, cada entidade de software) na fase de desenvolvimento para que este possa ser adaptado automaticamente sem a participação de seres humano (desenvolvedores). Como forma de avaliar nossa abordagem, estudos de caso foram conduzidos e os resultados mostram que esta pode contribuir de forma eficaz para a área de ES4Sa.

***Palavras-chave*:** Abordagem de desenvolvimento. Processo de adaptação automática. Software autoadaptativo.

Abstract: *The development of Self-adaptive Software (SaS) is a complex task, since this type of software constantly deals with structural and/or behavioral changes at runtime so that the needs of its users or its execution environment are met. Although important initiatives in the area of Software Engineering for SaS (SE4SaS), including approaches, processes, methods, and techniques for the SaS development, can be found in the literature. Approaches with automated support need to be explored, since they are an alternative to maximize the speed of SaS implementation and, at the same time, minimize the involvement of developers. Based on this scenario, this article presents an approach with automated support for the SaS development. It acts on a controlled adaptation modality, i.e., software engineers define the adaptation level supported by SaS in the development stage and the SaS can be automatically adapted without the participation of developers. Case studies were conducted for the evaluation of our approach. The results show that this approach can effectively contribute to the SE4SaS area.*

***Keywords*:** *Automatic adaptation process. Development approach. Self-adaptive software.*

1 Introduction

Over recent years, our society have increasingly become dependent on software systems, since they have acted in important segments, including public and private institutions, airports, communication systems, among others. Therefore, such systems must be prepared for their functions in normal operation and available 24/7 (i.e.,

[1]Department of Statistics, Applied Mathematics and Computation, Univ Estadual Paulista - UNESP, Rio Claro - SP - Brazil.
{frank@rc.unesp.br}
[2]Department of Computer Systems, University of São Paulo - USP, São Carlos - SP - Brazil.
{elisa@icmc.usp.br}

http://dx.doi.org/10.5335/rbca.2015.4224

24 hours per day, seven days per week). Moreover, most of them must to operate under adverse conditions, but maintain their integrity of execution. Features, such as robustness, reliability, scalability, and self-adaptation (self-star properties) are required by these systems. The two latter have stimulated the growth of a research area called Software Engineering for Self-adaptive Software (SE4SaS), whose object of study is the development of Self-adaptive Software (SaS). In short, this type of software incorporates new features (structural or behavioral) at runtime, i.e., adaptations with no interruption in the execution [1], [2] and [3]. According to Affonso & Nakagawa [4], Andersson et. al. [5], and Salehie & Tahvildari [6], software adaptation, when manually performed, becomes an onerous (regarding time, effort, and money) and error-prone activity, mainly due to involuntary injection of uncertainties by developers.

To overcome such adversities, efforts have been devoted to the establishment of approaches, processes, methods, and techniques that support the SaS development. Such efforts can be summarized into three phases: (i) treatment of requirements for SaS and their degree of uncertainties [7] and [8]; (ii) software engineering processes for self-adaptive systems [9] and [10]; and (iii) automated solutions for the adaptation of software systems at runtime [4], [11] and [12]. According to Lemos et. al. [10], Erradi et. al. [13] and Whitehead [14] automated approaches have been adopted, since they are an alternative to maximize the speed of SaS implementation and, at the same time, minimize involvement of the developers. Based on such a context, this article presents an Approach with Automated Support for SaS (AAS4SaS), which enables the development, monitoring, and adaptation of the software at runtime [4] and [11].

The AAS4SaS offers a set of guidelines to assist software engineers in the SaS development in a controlled adaptation modality. From this point onwards, SaS can be also referred to as software entities or simply entities, since they represent smaller software units in both stages development and adaptation. In short, software engineers define the adaptation level (structural and/or behavioral) supported by each software entity during the development phase. In the execution environment, these entities can be automatically adapted (automatic mechanisms) without the participation of developers [4], [11] and [15]. The process for automatic adaptation represents a comprehensive and complex solution that acts on the software adaptation at runtime. It comprises a set of mechanisms organized into a reference architecture composed of a core for adaptation and four additional modules namely: development, action plan, adaptation rules, and infrastructure [4]. Such modules use computational reflection [16], which is an important resource for inspection and adaptation of software [1]. A case study was conducted to prove the viability of our approach and the results can be considered an important contribution to the area of SE4SaS.

The remainder of the article is organized as follows: Section 2 addresses the background and related work; Section 3 presents the Approach with Automated Support for SaS; Section 4 describes a case study; finally, Section 5 addresses the conclusions and perspectives of future work.

2 Background and Related Work

This section presents the background (concepts and definitions) and related work that have contributed to the development of the present study. Concepts of reflection and adaptation techniques are described. Next, related work on approaches, methods, and techniques for the SaS development is addressed.

2.1 Background

Computational reflection, or simply reflection, can be defined as any activity performed by a system on itself, being very similar to human reflection. The main goal of software systems that present reflection is to obtain information about their own activities, so as to improve their performance, introduce new capabilities, or even solve their problems by choosing the best procedure. Therefore, the use of reflection enables software to be more flexible and susceptible to changes [1], [3], [5] and [13]. Next, some main uses of reflection for software adaptation are exemplified.

Borde et. al. [17] proposed a technique based on reflection for software components adaptation concerning structure and behavior. The existing functionalities are preserved and others (new requirements) are added, constituting a new software component. This technique also provides control over the number of modifications that can be applied to software components, as these components can quickly increase in size and their reusability can be

reduced. In Chen [18], reflection is used as a mechanism for the dynamic adaptation of distributed systems. The author proposed an RMI (Remote Method Invocation) extension, named XRMI (eXtended RMI) to monitor and manipulate remote invocations among components during a dynamic adaptation. This work is based on the Proxy pattern, which enables remote objects to behave in a transparent way for the application as if they were locally deployed. Tanter et al. [19] proposed a tool, named Reflex, for software dynamic adaptation based on reflection and Aspect-Oriented Programming (AOP). It provides both structural and behavioral facilities by adopting a reflection model, which enables a selective and fine-grained control of where and when reflections occur. Additionally, Janik & Zielinski [20] developed a technique named AAOP (Adaptable AOP), which is an AOP extension and adapts non-functional requirements. In short, it comprises a set of adaptable non-functional aspects integrated into the software at runtime, so that the functional requirements of the system remains intact. According to Peng et al. [21] and Shi et al. [22], reflection has been successfully used in the reuse of software components and implemented on a large scale in the reuse of software architecture and its components. The authors divided the architecture into two levels: (i) meta level, which contains the architectural components, i.e., information describing the base level as architecture topology, components, and connectors; and (ii) base level, which can be considered a traditional software architecture. Therefore, there are important initiatives for the use of reflection in the development of software systems.

2.2 Related Work

Uncertainties of requirements in SaS are a consensus among many authors and some initiatives have been proposed by different research groups. Esfahani & Malek [8] and Esfahani [23] described a debate about uncertainties in self-adaptive systems and highlighted their causes and effects. As a contribution to the research topic, they developed a framework for managing uncertainties in self-adaptive systems based on decision-making. Moreover, the decisions taken by this framework are supported by mathematical proofs. Souza et. al. [24] and [25] proposed a new type of requirement for self-adaptive systems called Awareness Requirement, which can be self-managed and/or manage other system requirements in relation to success or failure of execution. The authors developed a framework that provides qualitative adaptation to target systems based on information from their requirement models. This framework acts in a "closed loop" to monitor the behavior of systems by selecting a new system variant (goal). Its main characteristic is extensibility and enables different adaptation algorithms to be used depending on the availability and precision of information (input and output). Finally, Bencomo et. al. [7] proposed an approach that supports requirement reflection and makes requirements available as runtime objects. It approach enables software systems to be endowed with the ability to reason about, understand, explain and modify requirements at runtime.

The software engineering process for the SaS has received special attention from researchers. Andersson et. al [9] developed a software engineering process for SaS composed of a modeling approach and a framework for evaluation and based on the Software & Systems Process Engineering Meta-Model Specification (SPEM) [26]. Basically, models are created to show how a system is developed and such system can evolve. Moreover, can also promote better communication among software components, reuse, and reasoning, which, in the long-term, may be automated. To Brun et. al [27], feedback loops [28] that control self-adaptation must become first-class entities. Therefore, the following questions about **5W1H** model are relevant for this context [4], [6], [10] and [29]: 1)**W**hat will be adapted? 2)**W**here will the adaptation occur? 3)**W**ho will perform the adaptation? 4)**W**hen will the adaptation be applied? 5)**W**hy will it be adapted? 6)**H**ow will it be adapted?. Liu et. al. [30] also applied feedback loops to SaS, however they focused on the architectural design of the software system. Salehie & Tahvildari [31] proposed a Goal-Action-Attribute Model (GAAM) for the decision process in SaS. It uses an action selection mechanism based on the building of a light-weight and scalable runtime model for the selection of the appropriate adaptation action(s). Finally, Norha et. al. [32] discussed the design of SaS based on feedback loops and proposed a reference model for management of adaptation based on context (environment).

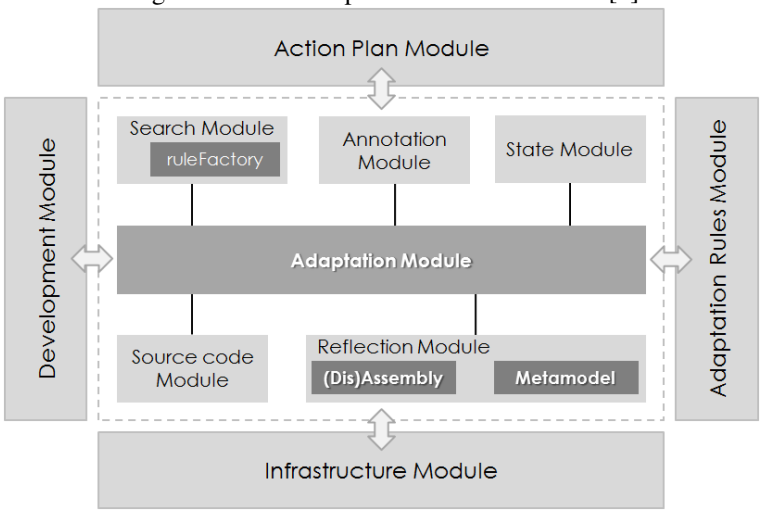
3 Approach with Automated Support for SaS

This section presents a reference architecture as an automatic process for the SaS adaptation. Initially, an overview of this architecture and its automatic process for adaptation is provided in Section 3.1. Next, the approach for the SaS development is briefly described in Section 3.2.

3.1 Reference Architecture for SaS

This section presents an overview of the Reference Architecture for SaS (RA4SaS). Figure 1 shows the architecture is composed of an adaptation core (dotted line) and four complementary modules: development, action plan, adaptation rule, and infrastructure. Following, a brief description of this architecture is presented [4].

Figure 1: General representation of RA4SaS [4]



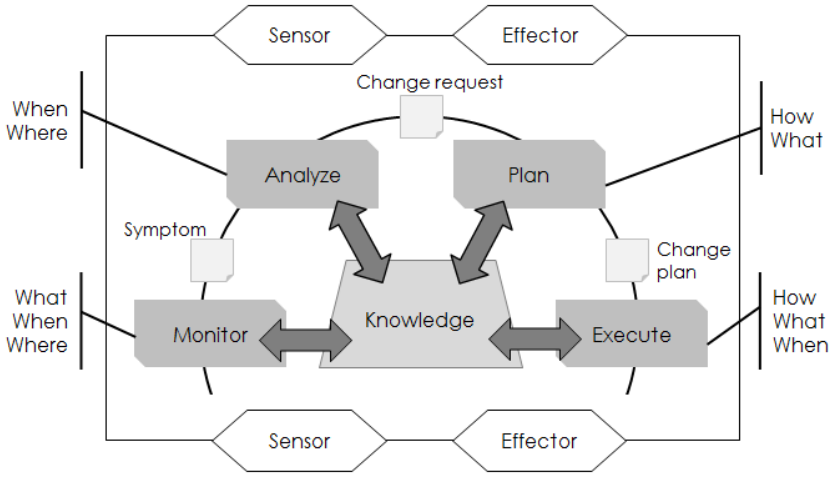
Development module. This module provides a guideline set for the development of software entities. The guidelines act on the following phases: requirement analysis, design, implementation, and evolution (adaptation). In short, the software entities are developed containing only attributes, constructors, and getters and setters methods. The development of these entities is addressed by architectural style in layer and each one is responsible for a set of system functionalities. This style aims to appropriately help to support the separation of concerns and increase the flexibility and maintainability of system software [33], in addition to facilitating automated support for adaptation [15], [21] and [22].

The evolution of self-adaptive systems can be considered a special issue, since it aims at capturing changes in a software entity during its life cycle. Initially, when an entity is developed, their goals are indirectly defined. Likewise, when an entity is adapted, its goals must also be modified. Therefore, disordered adaptations may make that an entity to quickly increases in size and its use may not be feasible in future systems. Thus, in order to manage on the adaptation number of the software entities and their goals, a metric base was adopted. Such metrics aim to guide the cohesion and granularity of the entities so that they do not lose their flexibility/capacity of adaptation. Details on the development approach are provided in Section 3.2.

Action plan module. This module assists the adaptation activity of the software entities. To do so, it must be able to control: dynamic behavior, individual reasons, and execution state in relation to the execution environment. We have adopted the MAPE-K (Monitor, Analyze, Plan, Execute, and knowledge) control loop as a solution to manage the software adaptation at runtime, since it has worked with the questions addressed in the 5W1H model [6], [28] and [34]. Figure 2 shows the relationship between MAPE-K control loop and 5W1H model.

In short, the *Monitor* process has mechanisms that collect data from sensors and convert them to behavioral patterns and symptoms. This process is related to **Where**, **When**, and **What** questions in SaS. The *Analyze* process correlates the data obtained to complex situations of the model. The autonomous systems can learn from the environment and predict future situations, which also helps to identify **When** and **Where** the modifications must be applied. The *Plan* process is responsible for creating **What** will be adapted and **How** to apply the changes in the software entity so as to achieve the best outcome. The *Execute* process must provide the mechanisms to execute the action plan established. This process is related to questions on **How**, **What**, and **When** to change. Finally, the *Sensors* and *Effectors* monitors are software entities that generate a data collection that reflects the system state and relies on in vivo mechanisms or autonomous subsystems to apply changes, respectively [6], [28] and [34].

Figure 2: MAPE-K control loop [28]



Adaptation rules module. This module is responsible for automatically extracting adaptation rules from the software entities. When an entity has been developed and inserted into the execution environment, a metamodel (reflection module) is instantiated. Next, an automatic mechanism (`ruleFactory`) is responsible for extracting this entity (metamodel) and creating a set of rules that describes its structure and behavior. Once generated, these rules are stored in repositories (rule base) and reused when a search for adaptation has been performed. Finally, these rules are written by the DROOLS framework [35].

Infrastructure module. This module provides support for the adaptation of the software entities at runtime. It comprises a set of mechanisms that act in various contexts. Among such mechanisms, three must be highlighted: (i) *dynamic compiling and dynamic loading*, which must replace the software entity at runtime without restarting the application or redeploying its components; (ii) **diagnosis of problems**, which must identify and report problems that occur when an adaptation activity is being performed; and (iii) *self-healing*, which must fix problems at runtime when an adaptation activity is being performed. For this, a framework based on learning techniques was designed by Leite et al. [36]. In short, when a software entity is developed and inserted into execution environment, this framework enables that a supervisor system is instantiated and coupled to such entity. The main purpose of this supervisor system is to collect data via sensors and send them to the classification module for identification of a problem. Next, when a problem is detected, one or more solutions may be presented in a list ordered by statistical measures. Details on this framework can be obtained in previous work these authors [36] and [37].

According to Salehie & Tahvildari [6], there are two approaches for the adaptation of software entities: (1) *internal*, when an entity can perform self-adaptation through mechanisms of the programming language - however it is limited, as it can not detect its context information; and (2) *external*, when a supervisor system (manager) interprets the context information and applies it to software entities. In general, only the second approach is used in the RA4SaS, since the software entities are monitored and adapted by external modules (subsystems).

Core of adaptation. This structure represents the "heart" of RA4SaS, since a set of modules manages the adaptation of the software entities at runtime. This core is organized into six modules: (i) search module, (ii) annotation module, (ii) state module, (iv) source code module, (v) reflection module, and (vi) adaptation module. The reflection and adaptation modules must be highlighted, since they act as software entities metarepresentation and adaptation "orchestrator", respectively. Next, each module is briefly described.

Search module. This module assists in the search for software entities in the execution environment when an adaptation activity has been invoked. Basically, it comprises two methods of search: (i) *semantic metamodel*, which can be defined as the concept description and notation set used to define a metamodel (reflection module). Such description represents structural and behavioral information of the software entities that are transformed into a metamodel of semantic relationship (Entity-Entity, Entity-Attribute, Entity-Method, or Entity-Attribute-Method); and (ii) technical information, which is specified only in technical information of systems software

(Entity, Attribute, or Method) in a template format. These templates are converted into as input parameters so as to search in the rule repository.

Annotation module. This module assists software engineers in defining the adaptation level of the software entities in the development phase. Therefore, it can be said that the software entities are developed in a modality of controlled adaptation since modifications (structural and/or behavioral) can be applied only at specified locations (i.e., where there are annotations). This module has three annotations: `@ClassAnnotation`, `@AtributteAnnotation`, and `@MethodAnnotation`. The `@ClassAnnotation` annotation represents the "main point" of the module, since it defines two important functionalities: adaptation level and entity type. The adaptation level determines how modifications can be applied (`CLASS`, `FIELDS`, `METHODS`, and `UNADAPTABLE`). When a software entity has been noted as `CLASS`, there must be annotations (`AtributteAnnotation` and `MethodAnnotation`) that determine the attributes and/or methods to be modified. When it has been noted as `FIELDS`, only the attributes (`AtributteAnnotation`) can be modified (added or removed). In this case, getters and setters methods also suffer the same changes, but they do not need to be noted. When it has been noted as `METHODS`, changes occur only in the methods (`MethodAnnotation`). Finally, when it has been noted as `UNADAPTABLE`, no modification in the software entity can be accomplished. The entity type determines the type of software entity to be adapted (`FUNCIONALITY`, `PERSISTENT`). A `FUNCIONALITY` entity represents a class composed of one or more methods that can be reused and transformed into either a remote method or a web service. A `PERSISTENT` entity represents a system's logical class endowed with persistence mechanisms (manual or automated frameworks) into a database. Finally, this module verifies whether the annotations have been correctly inserted, since without such information, the reflection module can not identify what will be adapted.

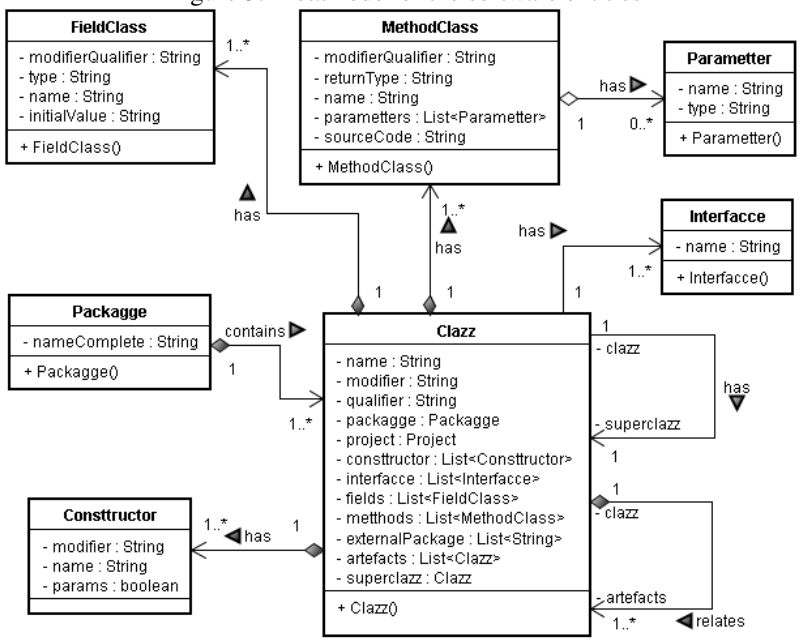
State module. This module preserves the current execution state of the software entities. When an entity has been selected for adaptation, the information contained in its current state must be preserved. The entity is then modified and its information is reinserted so that its execution is not interrupted. Basically, this module enables the conversion of an entity into a file (`.xml`) and vice versa. In short, the choice of XML (eXtensible Markup Language) for performing these operations is related to the following facilities: files handling (reading and writing), integration with different programming languages, and implementation facility.

Source code module. This module generates the source code of new software entities based on an instantiated metamodel by reflection module. To execute this operation, the software engineer must provide a software entity template based on the layered architectural style (e.g., logical, persistence, utilities). Basically, the module comprises three functionalities to generate the source code that meets the adaptation interests: (i) *structural*, when only one or a list of attributes should be added or removed from the entity - in this case, the getters and setters methods that manipulate these attributes are modified; (ii) *behavioral*, when only a list of methods should be added or removed from the entity; and (iii) *structural and behavioral*, when one and/or a list of attributes and methods should be added or removed from the entity. Finally, it has a mechanism for the version control of the entities source code that prevents them from being overlapped and keeps all versions of all stakeholders.

Reflection module. This module is organized into two submodules. The first assists in the "disassembly" and "assembly" of the software entities by using the annotation module for obtaining the adaptation level supported by each entity. Next, the disassembly process is then started, and structural and behavioral information is recovered via reflection and inserted into a metamodel, which is inside the second submodule. According to Systems and Software Engineering Vocabulary [38], a metamodel is logical information model that specifies the modeling elements used within another (or the same) modeling notation. After this metamodel has been instantiated, new information, according to adaptation interests, can be either added or removed, generating a new metamodel. This new metamodel is then transferred to the source code module so as to create the new software entities. Figure 3 shows the metamodel, which enables the adaptation of object-oriented software systems or make use of the structure of classes as units of software systems, as it can notice the presence of classes (`Clazz`), attributes (`FieldClass`) methods (`MethodClass`), etc. This strategy can be considered relevant for developers, since they can develop self-adaptive software entities in a notation to their expertise area.

Adaptation module. This module can be considered an RA4SaS "orchestrator", since it performs calls and coordinates all activities of the other modules (Core of adaptation). In a closer examination, it acts as a supervisor system of the software entities, monitoring their requisitions in the execution environment. To do so, it implements a well-defined process to adapt a software entity at runtime. In short, an entity is disassembled (metamodel), adapted (new information), and automatically reassembled as an "assembly line". In addition, an

Figure 3: Metamodel of the software entities



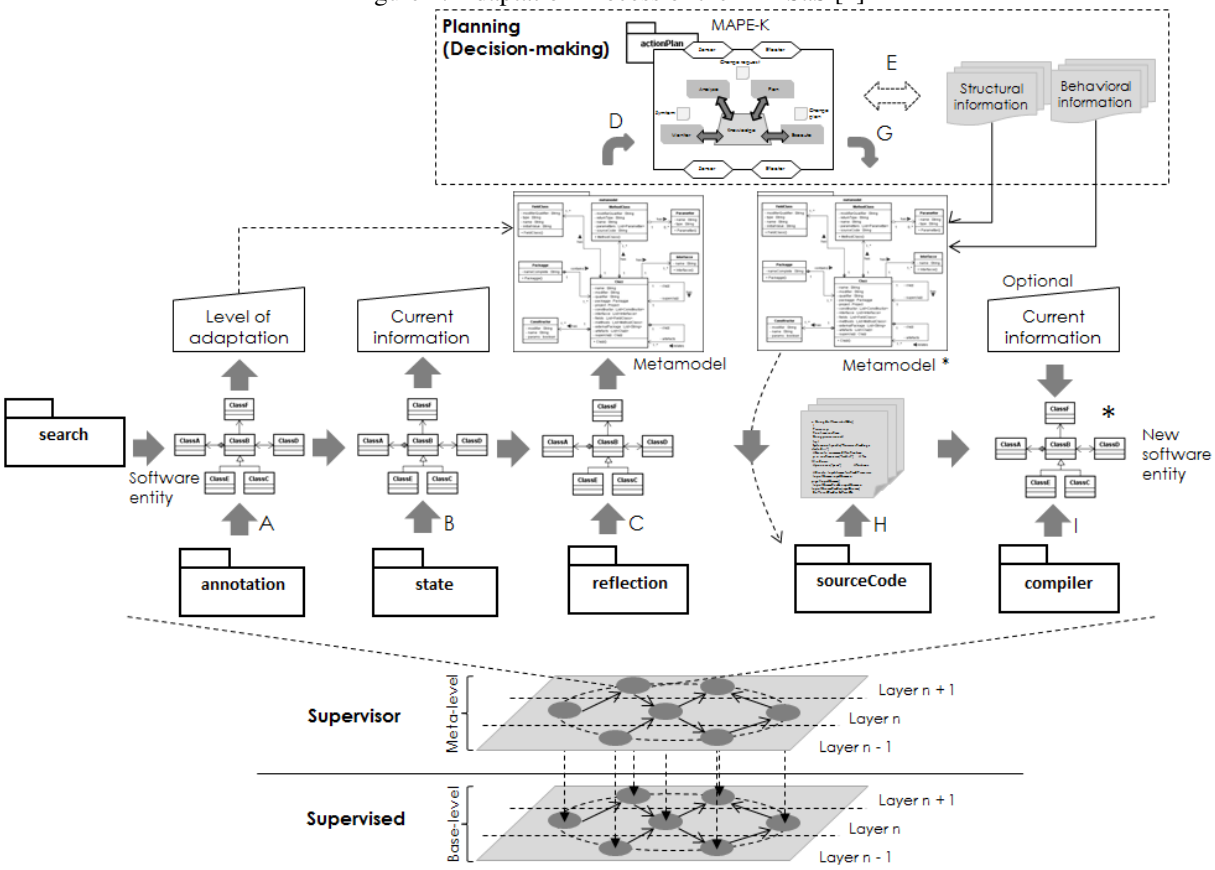
important feature for this module is the ability to interpret erroneous messages when a software entity is being dynamically compiled/loaded, since such messages are useful information for the corrections in the source code. Finally, it is important to emphasize that this process must be performed by software engineering tools, which reduces the implementation complexity and minimizes the generation of uncertainties. Details on the modules of the RA4SaS can be obtained in Affonso & Nakagawa [39].

3.1.1 Process for the SaS Adaptation

The RA4SaS has an automatic process for SaS adaptation which represents a logical step sequence (module orchestration) so that a software entity can be adapted. Such process is organized into nine steps, which are executed as an "assembly line", as illustrated in the Figure 4.

Initially (Step A), the adaptation level supported by each software entity is verified, since it will be used to instantiate the metamodel (Step C). Step B is responsible for preserving the current execution state of each software entity (during adaptation). This information will be reused in Step I if the software entity has not changed domain (goal). In Step C, the software entity is "disassembled" and a metamodel is instantiated. Such metamodel contains the structural and behavioral information of the software entity, as well as its adaptation level obtained in Step A. Step D basically consists in establishing an action plan for the entity adaptation. However, this is a complex and macro activity (dotted line), since it is composed of four steps (D, E, F, and G). The action plan should establish adaptation criteria so that the entity goals do not become unfeasible. The rule base must be used (Step E), since it provides guidelines regarding the feasibility of adaptation and goals. The adaptation procedure (action plan) must be established based on structural and behavioral requirements (Step E) and 5W1H model. A new metamodel is then generated (Step G) and will be transferred to the source code module. Finally, the source code of the software entity is generated (Step H) and compiled so that it can be inserted into the execution environment (Step I). In this last step, if the entities changes do not impact on the domain change (goals), the preserved information (Step B) is reinserted in this new entity to replace the old ones in the execution environment. The entities are transparently replaced for their stakeholders, since they have no perception of the changes in relation to the preservation of the execution status and the new generated instances.

Figure 4: Adaptation Process of the RA4SaS [4]



3.2 Approach for the SaS Development

The SaS development has specific characteristics in comparison to the traditional one, since this type of software enables changes to be incorporated at runtime. It also requires different approaches, an automatic process for adaptation, and flexible architectural models. Based on this context, this section presents an overview on SaS development focused on an approach (AAS4SaS) that represents the development module of the RA4SaS (Figure 1) [4] and [11]. Figure 5 shows the main phases of the AAS4SaS: analysis, design, implementation, test, and deployment.

Initially, software engineers and domain specialists must comprehend the problem based on requirement specification so that the development viability can be assessed (**Analysis phase**). Supported by the search module of RA4SaS, they search in the repositories of software entities so as to locate entities of identical or similar features. Identical entities can be reused immediately and dispense a development approach, just goals of such entities must be updated, since they can act in different context (domains). Similar entities undergo an evaluation process so that changes (requirements and goals) to be implemented can be validated. The states of off-line adaptation of the software entities are illustrated in Figure 6.

Preliminarily, a comprehension of changes so that an analysis regarding viability of these change can be performed (`Change Comprehension` and `Change Analysis` states). Such analysis (`Change Analysis` state) is supported by action plan module of RA4SaS. Next, the adaptation phase of this process is conducted (dotted line) so that the changes can be applied. In short, when a change activity is being performed `Change Implementation` state, two states can be assumed: (i) *Succeeded:* when a change has been successfully implemented and the activity is completed (final state), or (ii) *Failed:* when a failure has been detected. At this moment two transitions can be performed: (1) return to the analysis step for the reevaluation of the action plan and start of

Figure 5: Development Approach of SaS

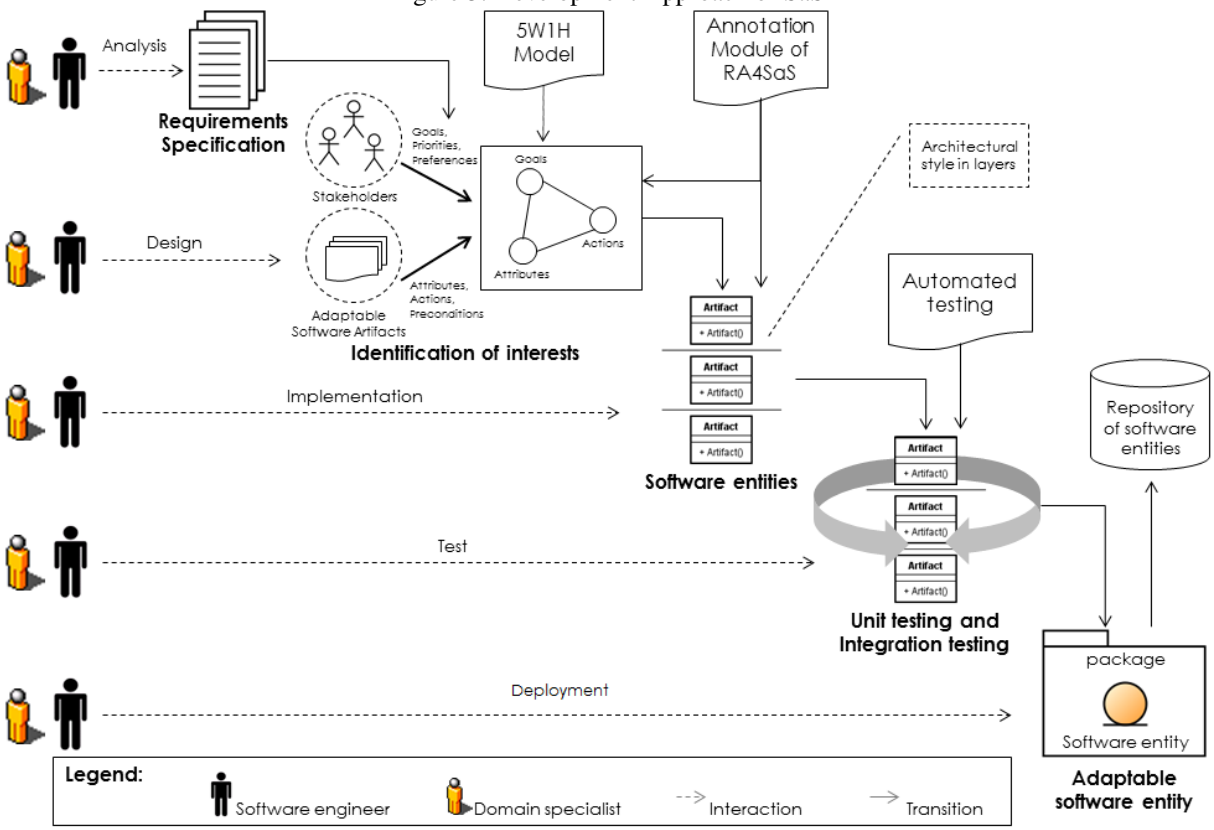
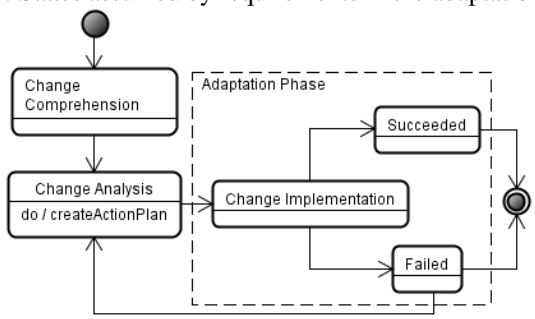


Figure 6: States assumed by requirements in the adaptation process



a new trial of adaptation, or (2) termination of the activity (final state), i.e., the design phase can be started.

In the **Design phase**, software engineers and domain specialists aim at identifying the adaptation interests. They make use of the 5W1H model and the RA4SaS annotation module concomitantly for the creation of a self-adaptive controlled environment. In short, the 5W1H model is composed of a question set that assists the users in the identifying and delimitating the system adaptation interests. Table 1 summarizes the questions of 5W1H and relates them to the possible action scopes for adaptation. After the system adaptation interests have been identified, annotations are inserted in each software entity so that they are automatically adapted without participation of humans [4] and [11]. The combination of 5W1H model with annotation module (RA4SaS) has shown an excellent solution, since it allows to establish Goals, Priorities, Preferences, Attributes, Actions, and Preconditions for SaS, both defined in GAAM (Goal-Action-Attribute Model) [31]. Finally, this combination enables a controlled adaptation modality with delimitations of scope and facility for the adoption of automatic mechanisms (RA4SaS

modules) for adaptation at runtime.

Table 1: The model 5W1H and adaptation scopes

Questions	Scopes
What *will be adapted?*	Attributes, software entities, architectural components, etc.
Where *will the adaptation occur?*	Software systems, software architectures, architectural components, etc
Who *will perform the adaptation?*	Autonomous systems, humans, both, etc
When *will the adaptation be applied?*	How often, anytime, constantly, etc
Why *will it be adapted?*	Errors in the project, new requirements (users or technology), etc
How *will it be adapted?*	Is there any action plan for adaptation? Has the adaptation been planned?, etc

In the **Implementation phase**, software engineers and domain specialists, based on the software entities designed in the previous phase, insert other meta-information in each annotation tag so that the implementation (codification) of the entities is started. The software engineers must provide an layered architectural style for the system development (i.e., each software entity). The architectural style is most commonly used in the implementation is the MVC (Model-View-Controller) model [40]. However, other architectural styles may be used or combined since they respect the constraints of organization presented in the design phase. In short, the architectural style must have four layers: presentation, middleware, application and persistence. The first is known as GUI (Graphical User Interface), or simply interface. The second is responsible for the communication between the interface and the system logic. However, it allows the creation of different distribution styles, such as client-server, service-oriented, and remote method invocation. The third is organized into two sub-layers: (i) system logic, which contains software entities, and (ii) system business logic, which contains persistence mechanisms of each entity and system business rules. Such mechanisms can be automatically implemented by the RA4SaS source code module. The business rules must be manually implemented by software engineers. Finally, the fourth layer represents the connectivity between the software system and the database.

Finally, in the **Test phase**, unit and integration tests are conducted in the with software entities developed (new) or adapted. As the source code of the software entities is generated by code generators (i.e., source code module of RA4SaS), the efforts devoted to this activity are minimized. After the tests, the software entities are packaged and inserted into the repository of entities software for to be reused (adapted) in future projects.

4 Case Study

This case study aims at exploring the applicability of the AAS4SaS guidelines in the development of software entities showing how they can be adapted at runtime by the process with automated support. For this, two types of modification were considered:

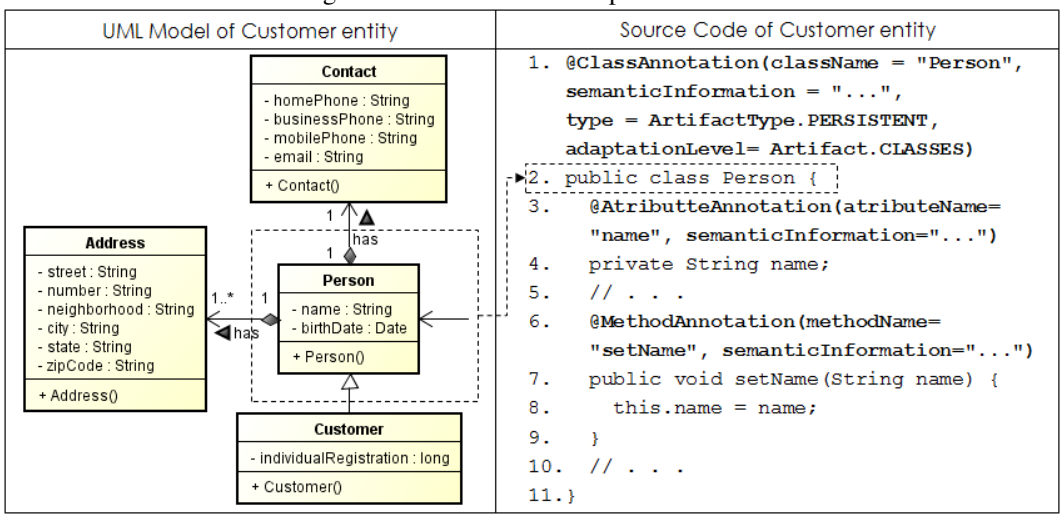
1. **Association of new functionalities**, which corresponds to the addition of new information (classes) into a selected software entity by aggregation, composition, or association;
2. **Extension of new functionalities**, which corresponds to the addition of new information (classes) into a selected software entity by inheritance relationship.

Before describing the case study, three technical considerations must be addressed. The **first** is related to the implementation of the RA4SaS (Section 3.1), which was instantiated in Java programming language [41]. For space reasons, the implementation details are not presented in this article. The **second** refers to the size and logic organization of the chosen system, which can be characterized as an information system for the bookstore

management. The chosen system allows us to show the guideline applicability (AAS4SaS) for the development of a software entity and adaptation at runtime (automated support - RA4SaS modules). The **third** refers to entity adaptation, which can be structural, behavioral, or both. For space reasons, a structural adaptation was chosen, since it can show the adaptation cases (1 and 2) of this case study. Moreover,

As presented in the Section 3.1, our RA works with a controlled adaptation approach, i.e., the software engineer must insert annotations in each software entity so that the automatic mechanisms in the environment execution can identify the adaptability levels of each entity. These levels contains parameters that determine where the new changes may be applied. Thus, when an entity is developed, an automatic mechanism performs a scan process to ensure that such annotations were correctly inserted. After validation process, these entities ca be stored in the entities repositories (execution environment) so that they may be invoked in future adaptations. Thus, for both the modification types (1 and 2), only the client software entity of the bookstore system, represented by `Customer` class (Figure 7) is considered. Based on the guidelines of the AAS4SaS, the `Person` class receives the `@ClassAnnotation` annotation as an adaptive software entity, since it has relationship with other system classes (`Address` and `Contact`). Line 1 shows the `Person` class annotated as `Artifact.CLASSES` adaptation level, which indicates this class can suffer structural and behavioral modifications. Therefore, it must have annotations for attributes (`@AtributteAnnotation`) and methods (`@MethodAnnotation`) - lines 3 and 6, respectively. Other relevant information provided in this annotation (line 1) is the artifact type (`ArtifactType.PERSISTENT`), which indicates that this entity (`Person` class) can be stored in a database.

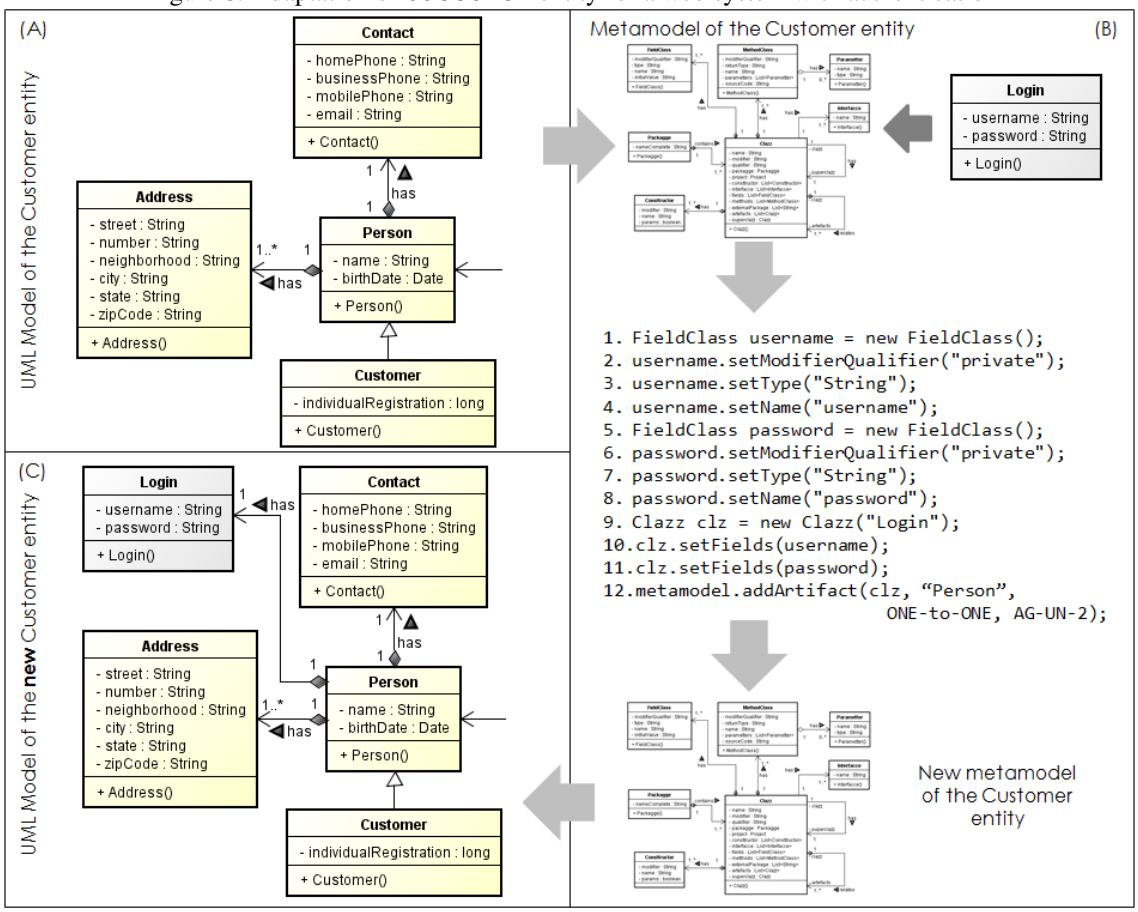
Figure 7: UML model of adaptation module



To show both types of adaptation, a preliminary step must be performed. This step represents the search of a software entity in the execution environment. In short, the software engineer or automatic mechanisms must provide the specification of a software entity (i.e., semantic information and input parameters) to the search module. Based on such information, this module must try to find in the environment a software entity with identical or similar features. When a identical entity is found, it may be automatically reused in other software system without adaptations. On the other hand, similar entities require adaptations in order to become the found entity compatible with the specifications provided by developers or automatic mechanisms. Thus, to show the first type of adaptation, the `Customer` entity (Figure 7), developed for a local system (bookstore), will be adapted so as to act in a web system with authentication (same domain), as illustrated in Figure 8.

Detailing the adaptation process of Figure 8, Square A shows the UML model of `Customer` entity being "disassembled" and a metamodel in `reflectManager.model` package (Figure 1) is instantiated (Square B) with structural and behavioral information on this entity. To do so, three steps of the process presented in Figure 4 were performed (A, B, and C). The first step retrieved the annotations of the `Person` entity, i.e., the adaptability level of this entity. In the second step, the current information of this entity was stored in a `.xml` file in order to maintain its execution state. Finally, the third step represents the instantiation of the metamodel per se. To meet the adaptation interests of the first case, i.e., for the `Person` entity can act in a web system with authentication,

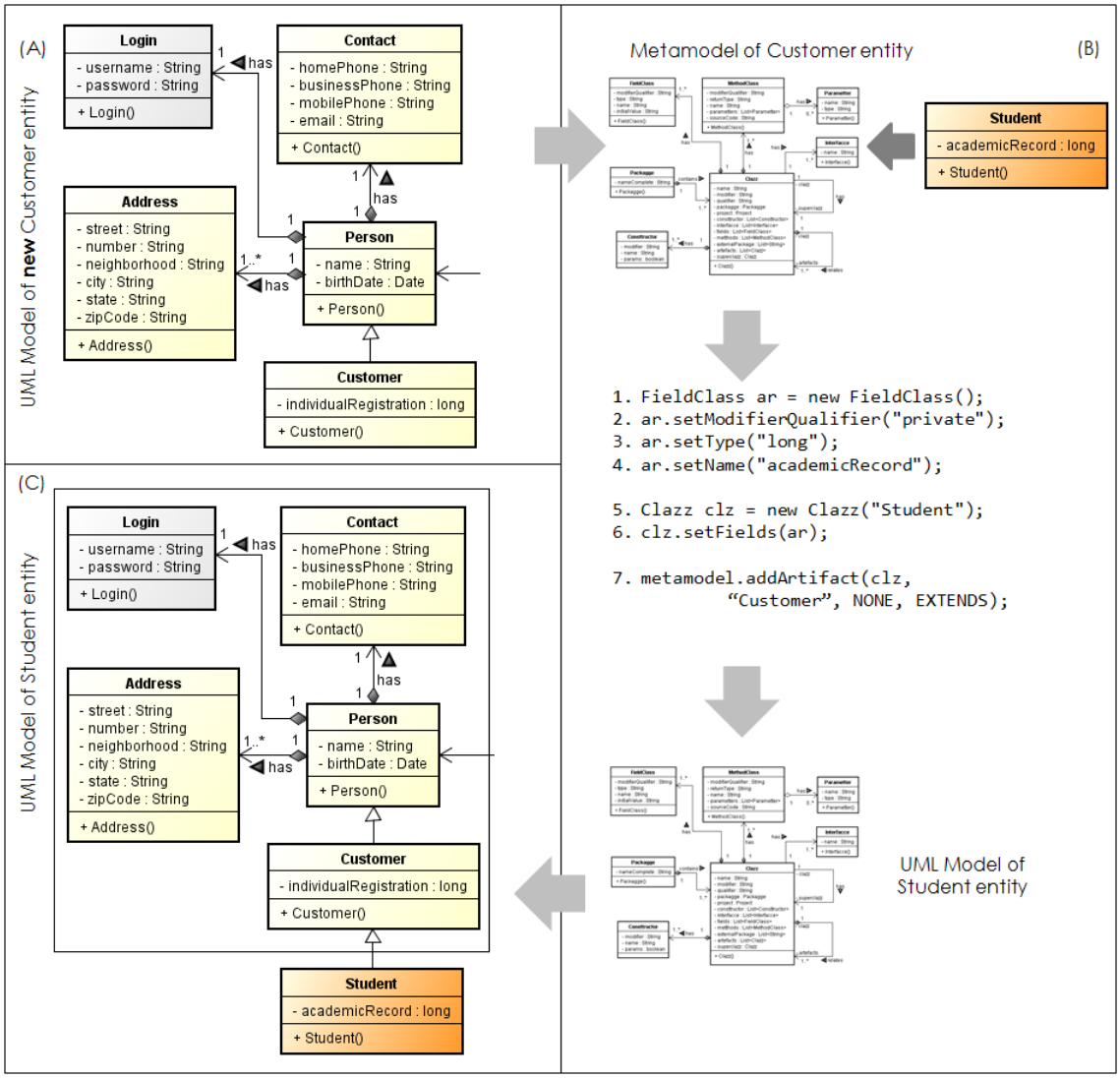
Figure 8: Adaptation of `Customer` entity for a web system with authentication



an software entity composed of two attributes (`username` and `password`) must be added to it as a possible solution. Therefore, a `Login` entity with such attributes must be created and associated (composition) with the `Person` entity (lines 1 to 12 - Square B). These lines represent a command set in Java programming language, which is the execution result of the steps D to G (adaptation process shown in Figure 4). Such steps represent the implementation of a control loop, which aims to monitor the requisitions in the execution environment and, in front of the change detection, recommend a solution set ordered by measures statistical (i.e., the most frequently used solutions) [36] and [37]. Lines 1 to 4 (Square B) show the command sequence for creation of the `username` attribute, such as access modifier, type, and name. The `password` attribute was created in a similar way (lines 5 to 8). The `Login` entity was created between lines 9 and 11. The constructor `Clazz ("Login")` in the line 9 defines the entity name in the source code and UML model. Lines 10 and 11 show the `username` and `password` attributes added to the `Login` entity. Finally, line 12 shows the creation of the relationship between the `Person` and `Login` entities. The first parameter (`clz`) represents the entity created (`Login`). The second (`"Person"`) and third (`ONE-to-ONE`) parameters indicate the entity name (metamodel) that will be associated with the new entity (first parameter) and cardinality between these entities, respectively. The fourth parameter (`AG-UN-2`) indicates the type of relationship (composition) and navigability (unidirectional) of the relationship (`Person` to `Login`). After the execution this command set (lines 1 to 12), a new metamodel is created and sent to the source code module for the generation of the new entity. The UML model (Square C) shows the new software entity (`Customer`).

To attend the second type of adaptation, the new `Customer` entity will be adapted to act in a school management system. This type of adaptation requires that this new entity operate in a different domain from which it was developed. Figure 9 shows the summarized steps of adaptation from `Customer` entity to `Student`. The adaptation process will not be presented at the same detailed level as the previous one (Figure 8), since the func-

Figure 9: Adaptation of `Customer` entity



tionality and execution of the RA4SaS modules area similar. Based on the `Customer` entity (Square A), a metamodel was instantiated with structural and behavioral information. For this entity to act in a school management system, a `Student` entity with `academicRecord` attribute must be created and added to `Customer` entity (Second lane). The creation of the `Student` entity is represented between lines 1 and 7 (Second lane). The `academicRecord` attribute is created between lines 1 to 4. In lines 5 to 6, this attribute is added to the `Clazz` class. Finally, line 7 shows the relationship between both entities (`Person` and `Student`). The first parameter (`clz`) represents the entity created. The second parameter (`"Customer"`) indicates the name of the entity (metamodel) that will be associated with the new entity (first parameter). The third (`NONE`) and fourth (`EXTENDS`) parameters show that there is no cardinality between such entities and the type of relationship (inheritance) between them, respectively. Finally, Square C shows the UML model of the `Student` entity.

5 Conclusions and Future Work

This article has presented a reference architecture (RA4SaS) and an approach (AAS4SaS) to support the development and adaptation of software systems at runtime. Software entities can be developed, transparently monitored, and adapted at runtime without the perception of the users and the involvement of humans. The AAS4SaS

guidelines enable the development of software entities for facilitating the adaptation by RA4SaS modules (automatic process), since these modules work in an assembly line, i.e., a software entity is automatically disassembled, adapted, and reassembled by the modules in a well-defined step sequence. Therefore, the main contributions of this article are:

1. To the area of SaS with a means that facilitates the development of systems with runtime adaptations;
2. To the area of reference architecture by proposing a architecture based on reflection that considers development and adaptations of software entities at runtime;
3. To the area of software automation, since our approach presents means of assembly line. The module of this architecture performs adaptation of software entities without intervention of the developers;
4. Finally, we believed that our approach (AAS4SaS) and reference architecture (RA4SaS) may be adequately used together with software development processes that have been used by companies, since both, reference architecture and process with automated support, seem to be complementary.

Some activities are being planned for future work:

- Case studies will be conducted for the evaluation of the approach and reference architecture presented in this article, since other type of adaptation of software entities must be investigated;
- Instantiate the reference architecture in other programming languages to evaluate its applicability and behavior in the adaptation of software entities;
- Apply the approach and reference architecture in the industry to evaluate the behavior of both when they are applied in larger environments of development and execution; and
- Evaluate the performance of the RA4SaS modules when a software entity is adapted, since the adaptation process of the RA4SaS is composed of task set that are sequentially executed. The results combined with our experience enable us to identify that some tasks can be executed in parallel, which certainly improve the adaptation time of the software entities, besides broadening the applicability of our approach (RA4SaS and AAS4SaS) to the domains of software systems whose time is considered as critical factor.

Therefore, we have a positive scenario of research, intending to become the RA4SaS and approach (AAS4SaS) effective contributions to the community of software development.

Acknowledgments

The authors would like to acknowledge PROPe/UNESP (Pro-rectory of Research / Univ Estadual Paulista) and Brazilian funding agencies (FAPESP, CNPq and CAPES) for the financial support provided to this research.

References

[1] MAES, P. Concepts and experiments in computational reflection. In: *OOPSLA 1987*. New York, NY, USA: ACM, 1987. (OOPSLA '87), p. 147–155. ISBN 0-89791-247-0.

[2] MORIN, B. et al. Models@ run.time to support dynamic adaptation. *Computer*, v. 42, n. 10, p. 44–51, oct. 2009. ISSN 0018-9162.

[3] VINOSKI, S. A time for reflection [software reflection]. *Internet Computing, IEEE*, v. 9, n. 1, p. 86 – 89, jan.-feb. 2005. ISSN 1089-7801.

[4] AFFONSO, F. J.; NAKAGAWA, E. Y. A reference architecture based on reflection for self-adaptive software. In: *Software Components, Architectures and Reuse (SBCARS), 2013 Seventh Brazilian Symposium on*. [S.l.: s.n.], 2013. p. 129–138. [In press].

[5] ANDERSSON, J. et al. Reflecting on self-adaptive software systems. In: *SEAMS/ICSE 2009*. [S.l.: s.n.], 2009. p. 38 –47.

[6] SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, ACM, New York, NY, USA, v. 4, n. 2, p. 1–42, maio 2009. ISSN 1556-4665.

[7] BENCOMO, N. et al. Requirements reflection: requirements as runtime entities. In: *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*. [S.l.: s.n.], 2010. v. 2, p. 199–202. ISSN 0270-5257.

[8] ESFAHANI, N.; MALEK, S. Uncertainty in self-adaptive software systems. In: LEMOS, R. et al. (Ed.). *Software Engineering for Self-Adaptive Systems II*. Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 7475). p. 214–238. ISBN 978-3-642-35812-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-35813-5_9>.

[9] ANDERSSON, J. et al. Software engineering processes for self-adaptive systems. In: LEMOS, R. et al. (Ed.). *Software Engineering for Self-Adaptive Systems II*. Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 7475). p. 51–75. ISBN 978-3-642-35812-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-35813-5_3>.

[10] LEMOS, R. de et al. Software engineering for self-adaptive systems: A second research roadmap. In: LEMOS, R. de et al. (Ed.). *Software Engineering for Self-Adaptive Systems II*. [S.l.]: Springer-Verlag, 2013. v. 7475, p. 1–32. ISBN 978-3-642-35813-5.

[11] AFFONSO, F. J.; RODRIGUES, E. L. L. A proposal of reference architecture for the reconfigurable software development. In: *SEKE 2012*. [S.l.: s.n.], 2012. p. 668–671.

[12] ELKHODARY, A.; ESFAHANI, N.; MALEK, S. Fusion: A framework for engineering self-tuning self-adaptive software systems. In: *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA: ACM, 2010. (FSE '10), p. 7–16. ISBN 978-1-60558-791-2.

[13] ERRADI, M.; BOCHMANN, G.; HAMID, I. Dynamic modifications of object-oriented specifications. In: *CompEuro '92*. [S.l.: s.n.], 1992. p. 654–659.

[14] WHITEHEAD, J. Collaboration in software engineering: A roadmap. In: *FOSE 2007*. Washington, DC, USA: IEEE Computer Society, 2007. p. 214–225. ISBN 0-7695-2829-5.

[15] BENCOMO, N.; BLAIR, G. Using architecture models to support the generation and operation of component-based adaptive systems. In: CHENG, B. H. et al. (Ed.). *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer-Verlag, 2009. cap. Using Architecture Models to Support the Generation and Operation of Component-Based Adaptive Systems, p. 183–200. ISBN 978-3-642-02160-2.

[16] ORACLE. *The Reflection API*. 2015. [*On-line*], *World Wide Web*. Avaliable at: <http://docs.oracle.com/javase/tutorial/reflect/TOC.html>, Last access: April, 2015.

[17] BORDE, E.; HAÏK, G.; PAUTET, L. Mode-based reconfiguration of critical software component architectures. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009. (DATE '09), p. 1160–1165. ISBN 978-3-9810801-5-5.

[18] CHEN, X. Extending rmi to support dynamic reconfiguration of distributed systems. In: *ICDCS 2002*. [S.l.: s.n.], 2002. p. 401 – 408. ISSN 1063-6927.

[19] TANTER, E. et al. Flexible metaprogramming and aop in java. *Sci. Comput. Program.*, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 72, n. 1-2, p. 22–30, jun. 2008. ISSN 0167-6423.

[20] JANIK, A.; ZIELINSKI, K. Aaop-based dynamically reconfigurable monitoring system. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 52, n. 4, p. 380–396, abr. 2010. ISSN 0950-5849.

[21] PENG, Y. et al. A reflective information model for reusing software architecture. In: *CCCM/ISECS 2008*. [S.l.: s.n.], 2008. v. 1, p. 270 –275.

[22] SHI, Y. et al. A reflection mechanism for reusing software architecture. In: *QSIC 2006*. [S.l.: s.n.], 2006. p. 235 –243. ISSN 1550-6002.

[23] ESFAHANI, N. A framework for managing uncertainty in self-adaptive software systems. In: *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*. [S.l.: s.n.], 2011. p. 646–650. ISSN 1938-4300.

[24] SOUZA, V. E. S. et al. Awareness requirements for adaptive systems. In: *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. New York, NY, USA: ACM, 2011. (SEAMS '11), p. 60–69. ISBN 978-1-4503-0575-4. Disponível em: <http://doi.acm.org/10.1145/1988008.1988018>.

[25] SOUZA, V. E. S.; LAPOUCHNIAN, A.; MYLOPOULOS, J. Requirements-driven qualitative adaptation. In: MEERSMAN, R. et al. (Ed.). *On the Move to Meaningful Internet Systems: OTM 2012*. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7565). p. 342–361. ISBN 978-3-642-33605-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-33606-5_21>.

[26] (OMG), O. M. G. *Software & Systems Process Engineering Meta-Model Specification (SPEM)*. 2008. On-line. Version 2.0. Avaliable at: <http://www.omg.org/spec/SPEM/2.0/>, Last access: April, 2015.

[27] BRUN, Y. et al. Engineering self-adaptive systems through feedback loops. In: CHENG, B. H. et al. (Ed.). *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer-Verlag, 2009. p. 48–70. ISBN 978-3-642-02160-2.

[28] IBM. *An architectural blueprint for autonomic computing*. 2005. On-line. White Paper, Third Edition. Available at: <http://www-03.ibm.com/autonomic/pdfs/ACBlueprintWhitePaperV7.pdf>, Last access: April, 2015.

[29] YANG, L. et al. 5w1h-based conceptual modeling framework for domain ontology and its application on stpo. In: *Semantics Knowledge and Grid (SKG), 2011 Seventh International Conference on*. [S.l.: s.n.], 2011. p. 203–206.

[30] LIU, C. et al. A problem oriented approach to modeling feedback loops for self-adaptive software systems. In: *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*. [S.l.: s.n.], 2012. v. 1, p. 440–445. ISSN 1530-1362.

[31] SALEHIE, M.; TAHVILDARI, L. Towards a goal-driven approach to action selection in self-adaptive software. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, v. 42, n. 2, p. 211–233, fev. 2012. ISSN 0038-0644.

[32] VILLEGAS, N. M.; MüLLER, H. A.; TAMURA, G. On designing self-adaptive software systems. *Revista Sistemas y Telemática (S&T)*, p. 29–51, 2011.

[33] BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice (2nd Edition)*. 2. ed. [S.l.]: Addison-Wesley Professional, 2003. ISBN 0321154959.

[34] DOBSON, S. et al. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, ACM, New York, NY, USA, v. 1, n. 2, p. 223–259, dez. 2006. ISSN 1556-4665.

[35] DROOLS. *DROOLS Business Rules Management System*. 2015. [*On-line*], *World Wide Web*. Avaliable at: <http://www.drools.org/>, Last access: April, 2015.

[36] LEITE, G.; AFFONSO, F. J.; NAKAGAWA, E. Y. *Projeto e Implementação de um Subsistema para Adaptação de Arquiteturas de Software utilizando o Mecanismo de Autocura*. [S.l.], 2015. (In Portuguese), Technical Report of Scientific Initiation: Projeto e Implementação de um Subsistema para Adaptação de Arquiteturas de Software utilizando o Mecanismo de Autocura, Last access: April, 2015.

[37] LEITE, G.; AFFONSO, F. J.; NAKAGAWA, E. Y. *A Framework for Self-healing Software Systems*. [S.l.], 2015. Technical Report: A Framework for Self-healing Software Systems, Last access: April, 2015.

[38] IEEE-VOCABULARY. Systems and software engineering – vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, p. 1–418, 2010.

[39] AFFONSO, F. J.; NAKAGAWA, E. Y. *A Reference Model based on Reflection*. [S.l.], 2015. Technical Report: A Reference Model based on Reflection, Last access: April, 2015.

[40] BUSCHMANN, F. et al. *Pattern-Oriented Software Architecture: a system of patterns*. [S.l.]: John Wiley and Sons, 1996. ISSN 0471958697.

[41] ORACLE. *Java Platform, Standard Edition (Java SE)*. 2015. On-line. Avaliable at: <http://www.oracle.com/technetwork/pt/java/javase/overview/index.html>, Last access: April, 2015.