PERVASIVE SYSTEMS DEVELOPMENT: A STEPWISE RULE-CENTRIC RIGOROUS SERVICE-ORIENTED ARCHITECTURAL APPROACH

Nasreddine Aoumeur[1] and Kamel barkaoui[2]

[1]Department of Computer Science, University of Leicester, LE1, 7RH, UK
[2]SYS:Equipe Systèmes Sûrs, Cedric/CNAM, France

ABSTRACT

To stay competitive in today's high market volatility and globalization, cross-organizational business information systems and processes are deemed to be knowledge-intensive (e.g. rule-centric), highly adaptive and context-aware, that is explicitly responding to their surrounding environment, user's preferences and sensing devices. Towards achieving these objectives in developing such applications, we put forwards in this paper a stepwise service-oriented approach that exhibits an explicit separation of concerns, that is, we first conceptualize the mandatory functionalities and then separately and explicitly consider the added-values of contextual concerns, which we then integrate at both the fine-grained activity-level and the coarse-grained process-level to reflect their intuitive business semantics. Secondly, the proposed approach is based on business rule-centric architectural techniques, with emphasis on Event-Conditions-Actions (ECA)-driven transient tailored and adaptive architectural connectors. As third benefit, for formal underpinnings towards rapid-prototyping and validation, we semantically interpret the approach into rewriting logic and its true-concurrent and reflective operational semantics governed by the intrinsic practical Maude language.

KEYWORDS

Context-awareness, ECA-Driven Rules, Architectural Connectors, Service-orientation, Adaptability, Maude Validation.

1. INTRODUCTION

Boosted by technological advances in networking, context-sensing and computation and pressed by stiff and global competitiveness, most of organizations are opportunistically joining their know-how into dynamic giant cross-organizational alliances. Striking features of any of such alliances, include: (1) *process-centricity*, that is, they exhibit very complex business processes with composing activities; (2) *high-agility*, where involved business processes and their activities / tasks are often governed by adaptive and evolving business rules [6,9,10,11,16]; (3) *context-dependency*, with user-preferences, adopted sensing devices and surrounding environment conditions as driving forces [4, 5, 28]; (4) *strong-dependability*, where the fatality of mal-functioning and failures may be economical und humanistic disastrous. Precise conceptualizations and formal techniques are thus highly required before investing any final deployment.

David C. Wyld et al. (Eds): NATP, ACSTY, CCCIOT, MLSC, ITCSS - 2022
pp. 115-132, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.120110

Towards reliably developing such complex, agile and context-dependent business applications, we are putting forwards in this paper a stepwise rule-based model-driven and context-aware architectural approach with the following software-engineering milestones:

[**Fine-grain Separation of concerns**]: We argue that for taming the complexity and agility of such business applications, involved concerns such as functionalities, context-awareness, security and quality need to be explicitly and separately handled at a first stage, then integrated at the architectural-level to reflect the reality of the application. Moreover, due to the intractable complexity of any business process, we suggest a fine-grained activity-centric handling of such concerns as first-class modelling entities.

[**Rule-centric architectural handling**]: Towards intuitively coping with agility in such volatile business applications, we are capitalizing on the ubiquity of business rules in such applications. Indeed, business rules reflect evolving policies and laws for doing / collaborating business [6, 9, 10, 16]. Furthermore, to enhance agility and bridge the gap to service-orientation, we shift any intuitive business rule towards transient architectural ECA-driven connectors [13]. More specifically, we propose ECA-generic patterns for both functionality- and context-awareness.

[**Rule-centric concurrent formalization**]: Towards soundly interpreting and validating this conceptualization, we further enforce to stay compliant with this rule-centricity. For that we propose Meseguer's rewriting (rule-based) logic [12] and its enabler efficient Maude language [3, 8].

[**Rule-centric service-oriented deployment**]: To preserve all strengths of the "business-foundation" phases, we propose rule-centric web-services for a compliant deployment [22]. However, to enhance readability and simplicity and keep with the space limitations this phase will not be further discussed. Detail about this phase will be addressed in the extended journal version.

The remaining sections are as follows. In the next section we summarize recent related work referring to any of the three milestones of our approach and their interleaving namely: Context-awareness, adaptability and its rule-centricity. The third section illustrates the working architecture of the proposed approach. In the fourth section, through a simplified banking process, we demonstrate how functionalities are captured at the architectural level using a tailored ECA-driven composition. In the fifth section, we present how context-aware knowledge is to be modelled at the architectural, with the introduction of the so-called context-intensive ECA-driven architectural connectors. In the sixth section, to reflect the intended intuitive business semantics of each activity, we present how both concerns require to be brought together around their activities within the concerned business process. We then address the formalization of the approach using rewriting logic and Maude language; Nevertheless, with the aim to boost the smooth readability of the paper we just skip the phase of the translation of ECA-driven connectors to the service-oriented RuleML [22]. We finally wrap up this paper with concluding remarks.

2. RELATED WORK: CONTEXT-AWARENESS, ADAPTABILITY AND RULE-CENTRICITY

As we pointed out in the introduction the innovative stepwise approach we are putting forwards for developing adaptive and context-aware business information systems bring together in a harmonious and architecturally-based manner, the following software-engineering ingredients: (1) Context-awareness; (2) Adaptability; (3) Separation in a rule-based way between context- and

functional concerns at the fine-grained activity-level; (4) Adoption of Service-orientation where the ECA-rules are captured as connectors between different service interfaces (5) Formalisation through the rule-based rewriting logic for rapid-prototyping and validation.

Saying that in the following we restrict ourselves to any related work that integrates at-least two of three of these software-ingredients; otherwise, the related work will transcend by far the space limitation of the paper.

The closest approach to our that integrates at-least the notion of rules and context-awareness in coping with dynamic and adaptive business processes in forwarded in [26]; furthermore, the paper addresses most of the related work in this respect and therefore we invite interested reader to go through these related works. In some detail, the authors propose to distinguish between internal and external contexts, both managed using what they refer to context-engine-, resources- and rule-managers. What seems to be close to our approach, is that the transition from one business activity to another is dynamically governed by the current context of the previous activity and the associated rules. Not mentioned in their paper compared to our is clearly the service-orientation and the formal-underpinnings.

Another recent interesting approach with some similarity to ours appeared in [29] and based on three-level architectural solution to develop context-aware application. The three-level are the *perception layer*, which covers in some sense the technical and technological-layer for sensing and interpreting the context; to mention here that in our approach this layer is assumed given at-priori as the technology is far advanced in this respect (see [14,16,27, 28]) for more detail concerning this technical side). The second level named *interference layer* and concerns the contextual rules, where the author suggests directly adopting the RuleML [22] XML-based low representation and not a business-level ECA-driven rules like our approach. Furthermore, besides the rules, the author proposes an inference-engine and context-brother to compute on these rules. the third-level is named the *application layer*; here the so-called application manager decides depending on the available context whether to run the suitable application. Finally, is to mention that a prototype called KoDA has been developed as a proof-of-concept for that approach.

Other approaches focussing mostly on the context-awareness and its modelling and implementation could be found among others found in [26, 27]. Concepts such as RFID, Ambient systems, Sensors and their classification are among others widely explained context-related ingredients there. Furthermore, the application of the context-awareness is experienced within different areas with the very interesting and critical health systems.

Finally, it is important to point out that our early ideas around separating location- from functionality-concerns, as preliminary work towards this currently more disciplined and stepwise approach for pervasive systems development, have been forwarded in [1]. That is, the present paper extends by far these ideas on several perspectives. Firstly, we are leveraging location-awareness with tailored context-awareness primitives to cope with pervasive systems. Second, we have been putting these first ideas into a throughout stepwise and service-oriented development approach. Third, as further contribution of this paper is the formalisation and validation of the approach using the rule-based rewriting logic and its intrinsic true-concurrent Maude Language. Last but not least, we are aiming to efficiently implement the approach using RuleML [10] and putting the approach into the context of development in particular of business information systems and processes at the fine-grained adaptability activity-level as will be detailed in the remaining sections.

3. MULTI-CONCERN AGILE SERVICE-ORIENTED BPS: APPROACH MILESTONES

As depicted in the Figure-1 below, the working general architecture of the approach we are pushing forwards can be highlighted as follows. Above all, we assume as given initial informal requirements such as: business goals and objectives, intentional business rules where specifically context-aware ones have been extracted from the environment sensing devices, actuators and so-on, informal business processes and their composing activities. Given such initial requirements, the gradual development of reliable context-aware service-oriented agile business applications, encompasses the following phases:

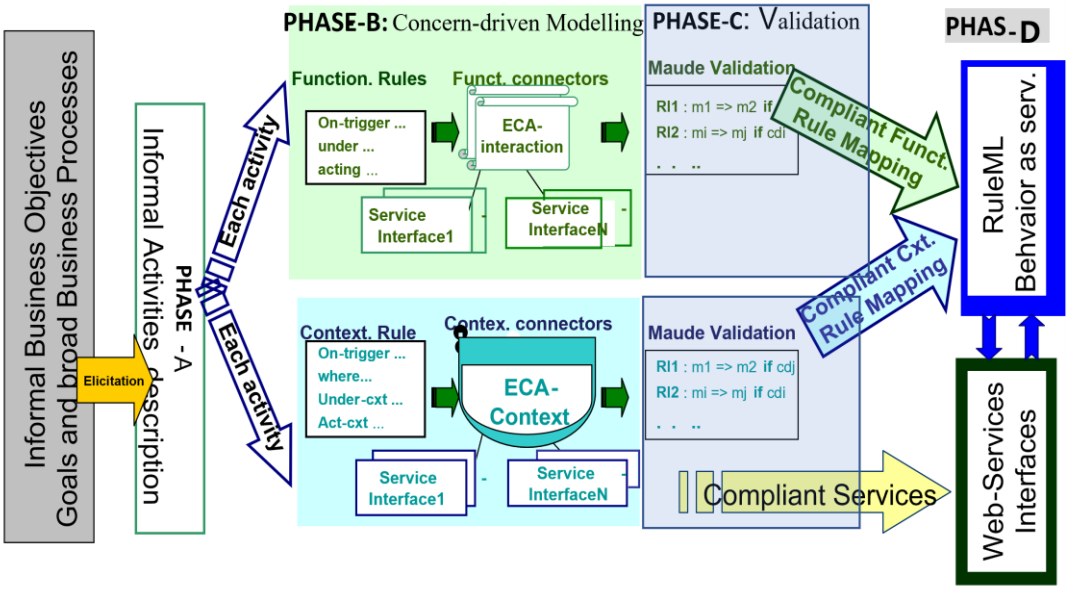

Figure 1. The forwarded Stepwise Architectural Context-aware Approach Milestones

[**Phase-A: Informal look at activities**]: The purpose of this phase is to re-visit the activities that may participate in any business process. For instance, we propose to list and describe them informally as preparation of the "business-conceptual" phase. In particular, we assume at this phase that context-aware information and knowledge have been already sensed at interpreted [27, 28]

[**Phase-B-Separated Rule-centric Modelling of Functionality and Context concerns**]: We consider this as *the most decisive phase* and as a distinguished capability of our approach with respect to the state-of-art. As depicted in Figure-1, this phase is progressive and involves two successive steps:

1. [*ECA-driven behaviour for any activity*]: That is, for each concern (e.g. functionalities, context-awareness), we separately describe the corresponding ECA-driven rules governing any activity. For the context-aware concerns, we propose a set of simple yet tailored business-level primitives to facilitate an intuitive description.
2. [*Architectural conceptualization using ECA-driven connectors*]: Towards deriving a disciplined and agile conceptualization while closing the gap to service-orientation, we propose to shift such informal ECA-driven descriptions towards architectural concepts.

We propose tailored ECA-driven architectural connectors, with roles playing service interfaces and their behavioural glues reflecting the composition logic [1].

[**Phase-C-Validation of Functionality and Context concerns using Rewriting Logic**]: For the formal validation, rapid-prototyping and verification, we propose yet another rule-based logic that completely fits within the proposed ECA-driven rule-based modelling phase. That is, this step within this "business-foundation"-level concerns the formal underpinnings of each concern at the activity-level using the true-concurrent rewriting logic-based semantics [12] supported by its Maude governing language [3,8].

[**Phase-D: RuleML-centric Service-oriented Deployment**]: At this ultimate phase, we propose to deploy the already certified and reliable service-driven application using Web-Services technology [21,25]. To stay compliant with the rule-centricity of the approach and thus preserve all its benefits, we take benefits from rule-based XML languages, specifically reactive RuleML [22], we leverage to service-orientation.

In the following sections, we will detail these phases, by considering a simplified case-study dealing E-banking. More precisely, as illustrated in Figure-2, we consider the following service-oriented business process, where after being identified, a customer can perform any banking action such a withdrawal, acquiring-loan, and so on.

- Customer identification and authentication.
- Customer performing a withdrawal (or deposits, loans, mortgages).

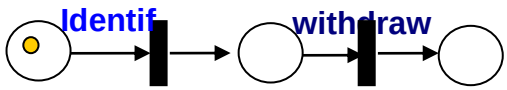

Figure 2. A Simple Petri-Net like Busines Process Model for basic banking Operations

We should already emphasize that most of existing approaches, as detailed in section 2, do not delve into the inside-behaviour of such activities composing a business process. Indeed, even service-oriented proposals, are restricted to only the modelling of message exchanges (e.g. send, receive and invoke) to perform an activity. In terms of Petri nets for instance as shown above, the activities are mostly considered as *black-boxes*, which opens a wide room for the programmers to implement them in ad-hoc and rigid if not in incorrect manner. The main goal of our approach consists thus in bringing to the conceptual level and precisely at the fine-grained activity-level at first stance as much *multi-dimensional knowledge* as possible in a manner that promotes adaptability, composability and dependability.

4. COMPOSITE SERVICE FUNCTIONALITIES AS RULE-CENTRIC INTERACTIONS

As we already point out, to capture the mandatory functionalities of any business activity, in each business process, we first propose to reformulate any governing intentional business rules into operational ECA-centric ones. Moreover, we endeavour describing such ECA-driven business rules at the interaction level, that is, we enforce ourselves to find out which business entities, modelled later as web services requiring from them appropriate interfaces, are involved in the associated ECA-driven rule. Then, the triggering events, the constraints to observe and the actions to perform are to be specified.

Afterwards, we propose to smoothly shift this interaction-driven informal business rules governing any activity into more disciplined architectural interconnections. For the architectural connector behaviour, which should reflect the ECA-driven rule, we propose a tailored ECA-driven generic pattern composed of the tailored primitives as to be described below.

4.1. Functionalities ECA-based Rules Illustrated and Intuitively Clarified

Let us straightaway consider the withdrawal action, indeed the identification-activity presents no functionality at-all as thus completely context-aware one as we will see it later, as a business activity in our banking business process. We propose to externalize at the interaction-level the rule governing the functioning of the withdrawal (i.e. in its simplicity (balance > amount-to-withdraw). Instead of speaking about the "withdrawal method", we are thus speaking about an agreement between the customer and (one of) his/her account(s) while banking. As direct benefit, we can now have different agreements depending on the profile of the customer (e.g. silver, golden) and its account (e.g. running, saving, asset). Moreover, we can address the policies of defining and adapting such agreements on-the-fly, and thereby increasing the competitiveness of any associated financial institution. Last but not least, the notion of triggering event (i.e. the customer wants to perform a withdrawal) is inherently to be understood now as an explicit "invitation" for the account to enter into composition with that customer.

Coincidently, these are the main features in the essence service paradigm SOC [19,25]. Firstly, SOC aims at dynamically composing of different partners (service interfaces) to achieve added-values, impossible to achieve by single partners. Second, SOC is based on service invocation using subscription and notification and dynamic binding.

As shown in Figure-3, for any withdrawal agreement, we require the following information from the two partners: From the customer, we require the triggering event and the fact that (s)he is owning the account; From the account, we require the balance and the debit operation, restricted to just the decreasing of the balance (i.e. NO internal conditions at all). Important requirement for the intended composition logic is the activity, as a composite service itself. We thus describe any activity-behaviour at-first level, based on the ECA-driven interaction puts in place to ensure the underlying business goal.

Example 1 (The ECA-based Functional rule for the Standard Withdrawal): As depicted below, the standard withdraw consists simply in externalizing the usual condition from the account component to the interaction level. The rule says that: *On the occurrence of a withdrawal event (subscription) from the customer, the targeted account balance should be greater than the requested withdrawal amount and in that case a debit message is (asynchronously) sent to that account to debit that account.*

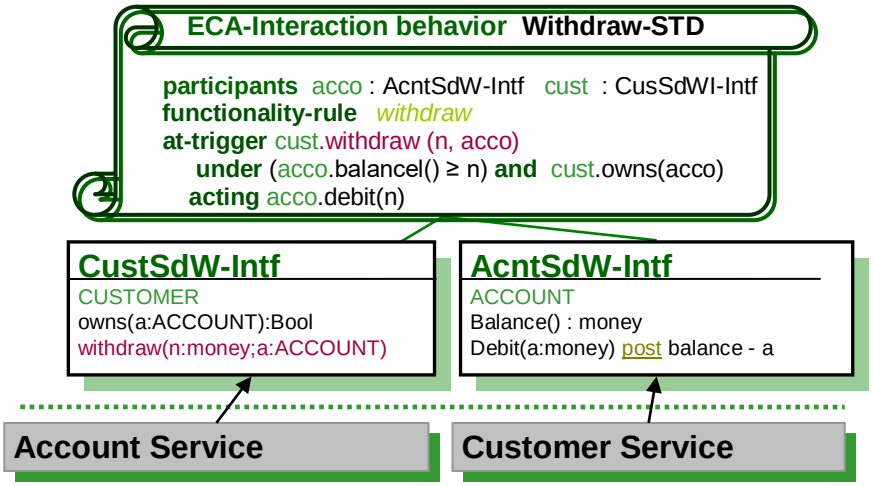

Figure 3. The Standard withdrawal ECA-rule as an Architectural Contract

Example 2 (The ECA-based Functional rule for the VIP Withdrawal): The second possible withdrawal agreement, as illustrated in Figure-4, consists in endowing "privileged" customers with a credit so that they can withdraw below their account balances. The interaction ECA-based rule as an architectural connector takes the following form, where all what changes in respect to the standard case is the condition that becomes more flexible.

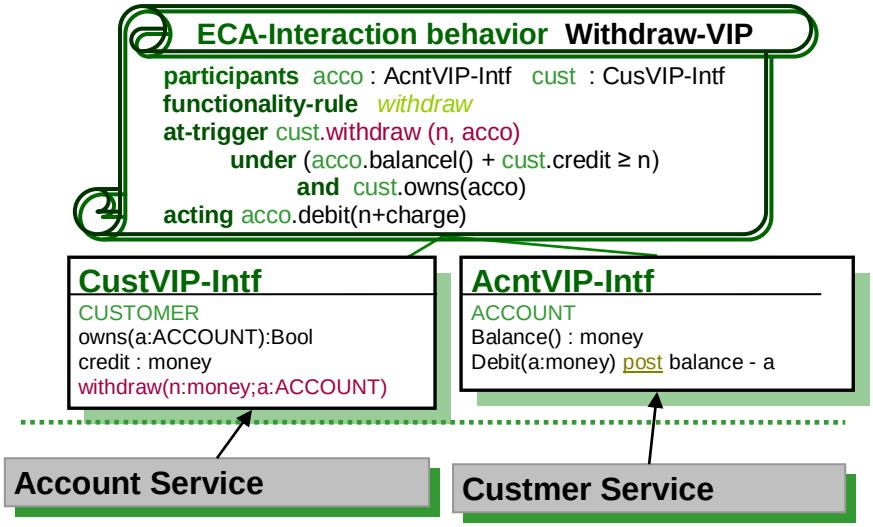

Figure 4. The VIP withdrawal ECA-rule as an Architectural Contract

Notice that a different partner is now required to play the role of the customer: we need a service that offers an operation for obtaining the credit limit currently assigned to the customer. Please note that, participants can be hierarchically organized with, for instance, silver and golden accounts as well as saving, asset accounts. For simplicity, we addressed just the simple flat case.

5. CONTEXT-AWARE CONCERNS AS TAILORED ECA-DRIVEN RULES

As we emphasized, the main objective of this contribution is to first explicitly separately address different concerns while tackling any context-aware service-oriented business applications. In the previous section, we demonstrated how interaction-centric functionality concerns can be conceptualized as transient ECA-driven connectors. In this section, we similarly present how context-aware concerns need to be extracted from any activity and modelled as tailored ECA-driven "contextual" connectors.

Towards forwarding suitable conceptual primitives for context-awareness and in contrast to functionality concerns, we first require contextual predicates [6] for reasoning about the surrounding environment and any involved devices and so on. In this contribution, we restrict ourselves to the role of *locations in defining* the context-aware behavior governing any activity. Nevertheless, as the reader may easily infer, the approach is flexible enough to be extended with further predicates to cope with device resources and user preferences, such as GPS when banking identification is performed using his/her Mobile device and cameras implanted at the ATM for face-recognition and/or fraud detection for instance. More precisely, we propose three context-aware predicates to allow reasoning about context-awareness in terms of ECA-driven rules.

- **The communication status**, reflecting the presence, absence, or quality of the link between locations where given services are performed but require exchange of knowledge (e.g. data, message). This is, captured through the "connect" predicate CNT:set(LOC) → BOOL. Where LOC stands any location-dependent (concrete or abstract) entity.
- The ability to continue the execution of an activity at another location, which requires that the new location is *reachable* from the present one so that the execution context can be moved. The construct *RC:LOC x LOC → BOOL* is proposed. It informs whether a given location is reachable from another one.
- The spatial relationship between two locations so that triggering events may be initiated. We abstract such predicate as: *Near2(LOC,LOC) → BOOL*.

As already mentioned, In the same manner other contextual predicates can be forwarded for informing and reasoning about resources such as devices memory, display, GPS-aware devices, processor-capabilities and cameras among others. We have introduced the three above just for illustration, but we are working on a complete set of primitives for reasoning about different context-aware situations. Similar primitives can be found in [6] and [14] among others.

As for interaction concerns, we propose *ECA-driven context* rules as conceptual architectural connectors to cope with the context-awareness, through the explicit use of the above contextual predicates. More precisely, to be compliant with the followed event-driven paradigm, we let unchanged the triggering primitive, that is, any context-aware rule will start like the interaction rule with the *at-trigger* primitive. To emphasize that now the constraints to be involved for any context-aware involve one or more contextual primitives (to test the current surrounding environment), we propose to split the condition part into two parts. A first part starting with where concerns the testing of the status of surrounding context. The second part concerns the usual conditions, though now dealing with context-awareness issues; we introduce such conditions by starting with under-cxt. Finally, the actions to perform are to be prefixed by act-cxt.

5.1. Context-awareness Concerns graphically illustrated and explained

In the following, we illustrate these context-aware concerns by considering the same activities we addressed at the functionality concern. More precisely, we first consider the withdrawal then the identification activities (which is fully context-dependent).

But before detailing that context concerns, let us again motivate more on the explicit and strict generic separation of these two concerns (i.e. functionalities and context-awareness) while modeling business activities. As depicted in Figure-5 and still with respect to the banking application, we have two strict yet complementary concerns while describing any activity of this application. That is, on the left hand-side, the behavioral functionality issues are to be expressed at the interaction level using (ordinary) business entities such as: (different kinds) customers and accounts.

On the right-side, while we always stick to the modeling at the composition-level to promote adaptability, the entities coming into play are more intrinsic contextual-aware entities: such as ATM, INTERNET, CARD, and so forth. These entities are at-least location-aware, where the context primitives influence by part the interaction.

Finally, it is worth pointing out that these two concerns are to be semantically related to reflect any activity as will be seen later. For instance, we will be speaking about *customer@atm*, *customer@internet*, *account@bank*, *card@atm* and so forth. That is, the business entities will be using or residing in associated context-aware entities. We finally, point out that such separation of entities has been recently reported in [29] as we have already detailed in the related work, although without emphasis on the composition as first-class neither at the fine-grained activity-level.

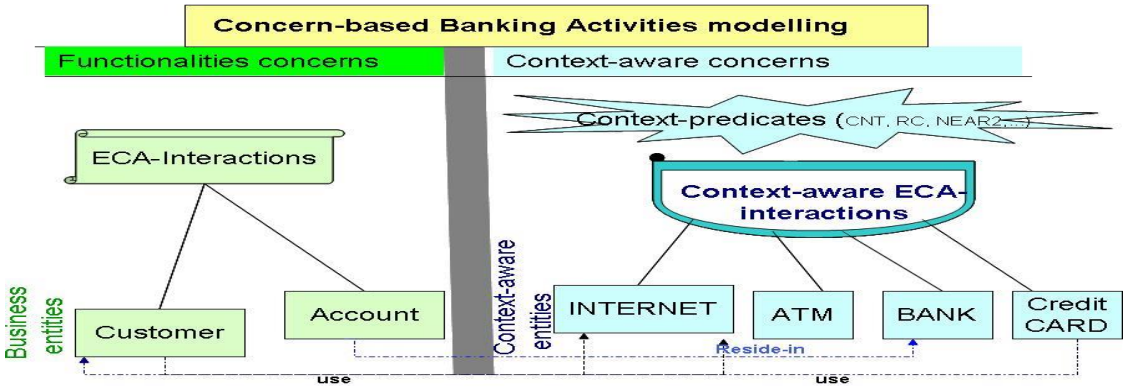

Figure 5. Graphically illustrated how the ECA-based Functionality-Concerns may Interact with the Context Ones for the banking Example

Now coming back to the specific activities in our simplified banking business process, that are, the *identification* and the *withdrawal* activities. Let us start with the withdrawal activity, we already discussed its behavioural functionalities at the interaction-level. Indeed, when we described these ECA-driven functionalities, we did not mention at all what are the contextual situations governing this activity. We were not concerned with the adopted business *channels* as described in the right-hand side of Figure-5 nor with *where* such withdrawal was taking place.

Example 3 (Context concerns for the Withdraw Activity): The conceptualization as shown in Figure-6 details the behavioural added-values and / or restrictions to be observed when baking at an ATM. Indeed, first as involved context-aware entities, we should have the ATM and BANK. From the ATM, we implicitly require its location but also properties such as the available cash and the default amount, both as hidden. An event for triggering the withdrawal is further required; but from the context-aware perspective, we do not care whether it is initiated automatically or from the user by pressing a specific button. Finally, the ATM is to be able to deliver money when the following constraints hold. First, as the first rule details, if the ATM is in a full connection with the corresponding BANK, the withdrawal is performed with the requested amount unless it surpasses the agreed-on maximum to withdraw or no enough cash is available. The second rule in contrast concerns the case where no connection is available (i.e. CNT(ATM, BANK) is false). In this case, only a default amount is allowed using the conditions that such ATM is endowed with off-line reachability to report on the performed transaction later (i.e. RC(ATM, BANK) is true). That is, the transaction is to be moved or migrated later using a banking operation such mv(wdr-op(atm-internal), bank).

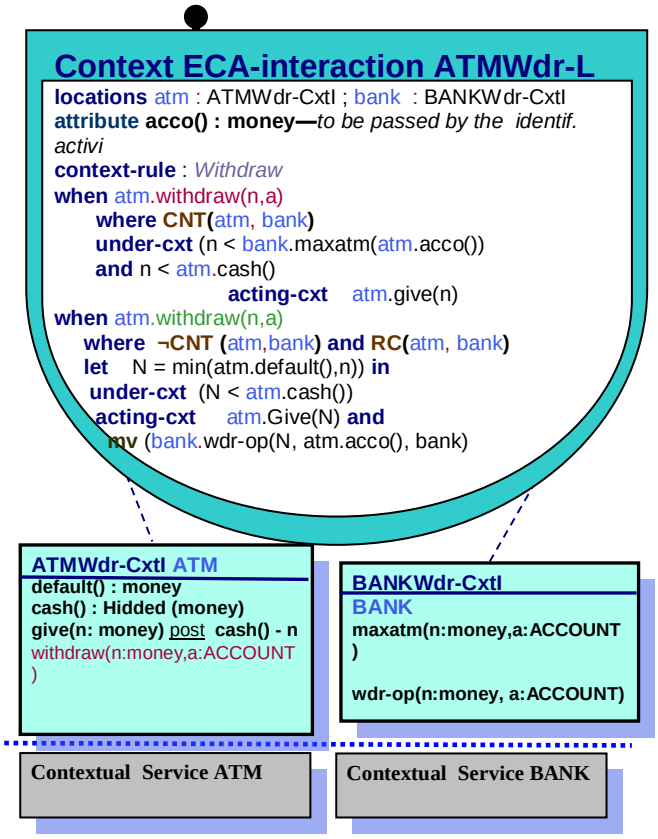

Figure 6: The ECA-based architectural Context-aware Rule for the Withdraw Activity

Example 4 (Context concerns for the Identification Activity): With the aim to illustrate more the crucial importance of context-awareness concerns, let us consider now the identification activity in any banking business process which in our case must precede the withdrawal activity. From the functionality's perspective, we did not skip it, instead there has been *nothing* to functionally describe for this identification activity. In other words, the identification is purely context-aware activity as it concerns the exclusive interaction of context-aware entities such as: INTERNET, PDA, ATM, CARD and where the connections and the other contextual primitives are decisive in defining the associated behaviour. As detailed in Figure-6, for the case of banking

at the ATM, that is, the identification via a Bank-CARD, we require from the ATM the ability to accept /(r)eject Bank-Cards and to enter Pins. We note that the accept message records the account and customer when successful. From the CARD, the hidden coded should present as well as the acceptance and rejection. We also require that the account number and the customer ids to be offered from the Card-magnetic. The corresponding ECA-driven interaction-rule says that we have first to enter the triggering event *EnterPin(Num)*. If the ATM reacts to that CARD, that is either the CARD is inserted or simply it is just near it (via infrared or Bluetooth connection), the entered code is checked with the stored card-code. Then, three attempts are allowed as possible ATM capabilities. That is, even this ATM withdrawal rule could have several variants reflecting specific ATM capabilities and Card (holder) specificities.

We similarly note that, the identification could be done via Internet / PDA. In this case we must enter password under the constraints that a LAN or WLAN connection is available.

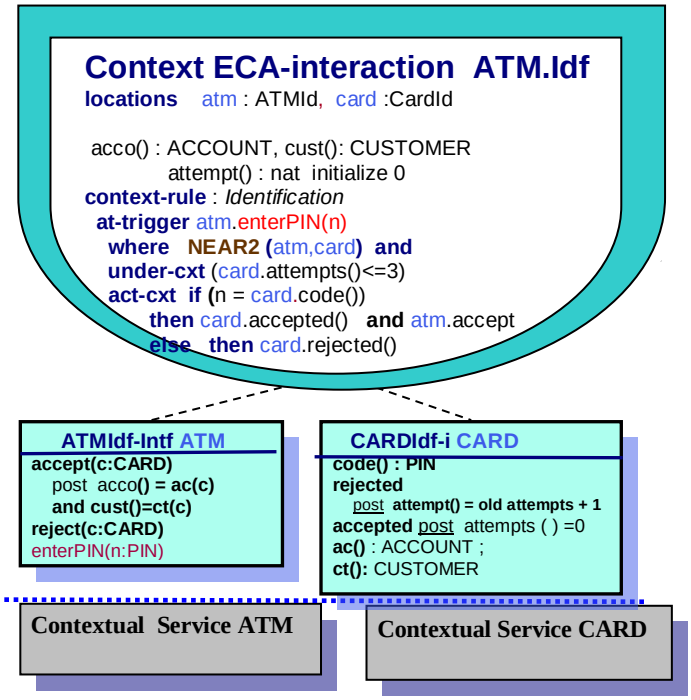

Figure 7. The ECA-based architectural Context-aware Rule for the Identification Activity

6. INTEGRATION OF CONCERNS: ACTIVITIES AND PROCESS MODELLING REFERENCES

In the two previous sections, we separately dealt with functionalities and context-awareness while modeling any activity as highly adaptive and behavioral tailored ECA-driven interactions. In this sense, both concerns can be modeled, evolved and reasoning about completely in separate manner, enhancing thus the mastering the application complexity and evolution.

Nevertheless, once such concerns and with respect to any involved business activity, in each service-driven business application are developed, we require bring them together to reflect the complete and intuitive business semantics we aim for any business activity. Being able to model these concerns separately does not thus mean that they are independent. The way a business activity is performed within a process system emerges from the functionalities as well as from the contextual ECA-driven rules that jointly apply to that activity. Indeed, whereas and still at the

conceptual-level the "What" question reflects the functionality concerns, the "Where" with its "How" capabilities (such as sensors, actuators, cameras, etc.) should reflect the context-awareness of any activity.

6.1. The Intrinsic Integration of functionalities and Context-awareness graphically illustrated and explained

To be more illustrative, when banking at ATM for instance, we have on the one side the withdrawal functionalities reflecting the intended interaction of the customer with its customer to perform the right withdrawal. On the other side, we have the added-value of opting for a withdrawal using the ATM, that is, the context-aware interactions while withdrawing.

As illustrated in Figure-7, with respect to this withdrawal activity, it is more logical at the end to speak about *customer@atm* and *account@bank*. That is, we have to bring together the functionalities and context-awareness ECA-driven rules together while running any withdrawal activity using the ATM. As shown in the picture, for different customers and ATMs, we may have different instances running each with the right functionalities and context-aware rules. Since that we were coherently using ECA-driven rules, this integration of concerns around activities is not that much difficult and become very intuitive. Indeed, we have just to join together different clauses as conjunction. More precisely, first the events require to be unified by integrating all their parameters. Then, all conditions in both selected rules have to gathered as conjunction. Finally, all actions in both selected rules have to performed.

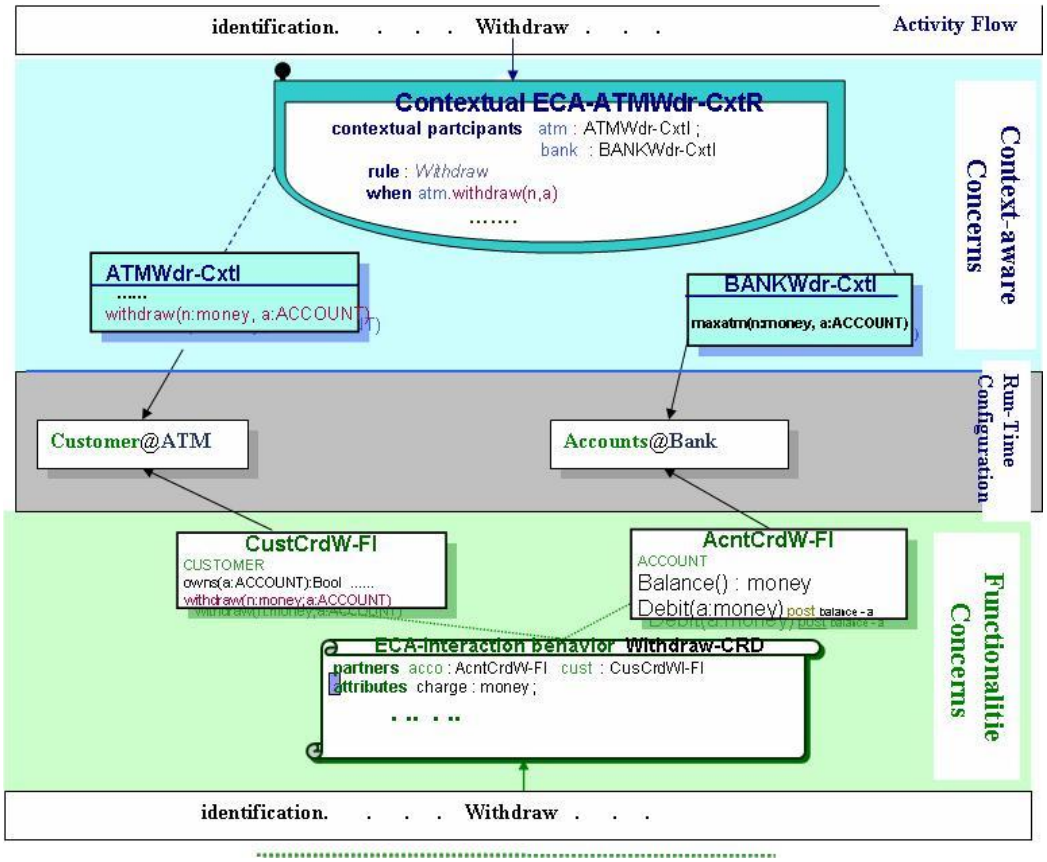

Figure 8. The Architectural Integration of the both Functional and Context-based Concerns Graphically Explained for the Withdrawal activity

Example 5 (Concerns Integration for the Withdraw Activity): The integrated rule of a withdrawal at ATM with customer enjoying a credit withdrawal could thus be represented as detailed in Figure-8. That is, first we have to unify the withdrawal triggering to become cust@atm.withdraw(cs, m). Then, we have to check that a connection between the ATM and corresponding Bank is holding (if not the integration concern the second context-aware rule of the withdrawal as given above). First, we have to check that the functionality rule holds, that is, there is enough money in the account plus the credit and that the customer is owning that account

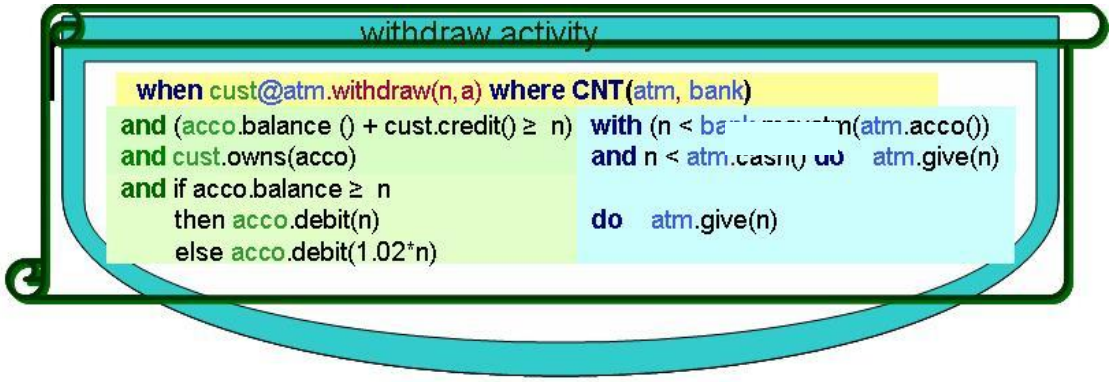

Figure 9. The Integration of the Functional and Context ECA-Driven Rules for the Withdrawal Activity

On the ATM context-aware side, we have to verify that the customer is allowed to withdraw such amount from the ATM and there is enough cash in that ATM. When all these constraints are holding, the account is debited, and the ATM deliver that requested amount.

Example 6 (Concerns Integration for the Identification Activity): We stress again that the integration of the identification activity of both concerns remains purely context-aware as no functionalities are bounded to that activity, nevertheless, we have to adapt the previous context-aware ECA-rule so that, for instance, the trigger atm.enterPIN(n) has to be changed to cust@atm.enterPIN(n), the condition from **NEAR2 (**atm,card**)** to **NEAR2 (cust@**atm,card@atm**)** and so on. That is the identification activity should look like as reflected in Figure-9 below.

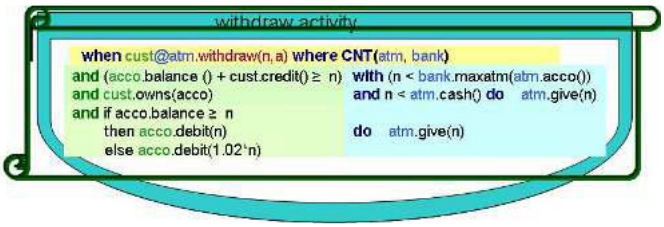

Figure 10. The Integration of the (Functional and) Context ECA-Driven Rules for the Identification Activity

7. FORMAL VALIDATION USING MAUDE AND REWRITING LOGIC

The semantical framework we are proposing for this new concern-based ECA-compliant service oriented architectural conceptualization is based on rewriting logic [Mes92], which has been proved very appropriate for dealing with concurrent systems. Further strengths making this logic very practical is the current implementation of the MAUDE language [3, 8]. In MAUDE object states in are conceived as terms — precisely as tuples— of the form *<Id : C|at1 : v1, .., atk :*

vki>. In this tuple : Id stands for object identity; C identifies an object class; and *at1, ..., atk* denote attribute identifiers with *v1, ..., vk* as current values. Messages (i.e. method invocation) are regarded as operations sent or received by objects, and their generic sort is denoted Msg. Object and message instances flow together in the so-called configuration, which is a multiset, w.r.t. an associative commutative operator denoted by ' ', of messages and (a set of) objects. The effect of messages on objects is captured by appropriate rewrite rules.

Example 7 (The validation of the Withdrawal Rules Using Maude): Without delving into detail about how to we systematically allow deriving the rewrite rules from the ECA-driven interactions of both concerns, we sketch here directly the Maude code corresponding to the withdrawal case, which should look like this module as illustrated in Figure-9.

```
omod WithdrawalAway is
  protecting Money .
  sub-sorts CustId   AcntId < OID .
  ***** participants
  class Account | Bal :  Money.
  class Customer | Own :  CustId .
  event Withdraw :  CustId  Money → Events
  msg-loc Debit :  AcntId  Money → Msg .
  Vars M, Max.  , Charge :  Money .
```

$$[\texttt{WdrAwy}]: Withdraw(Cs, M)\ \langle Ac : AcntId | Bal(Ac) \rangle\ \langle Cs : CustId | Own(Cs, Ac) : True \rangle \Rightarrow \langle Ac : AcntId | Bal(AcntId) \rangle \langle Cs : CustId | Own(Cs, Ac) : True \rangle debit(Ac, M + Charge)\ \texttt{if}\ (M > Max.) \wedge (Bal(Ac) > M)$$

Figure 9. The Validation of the Withdrawal Rules using the Maude Language

Then using the current implementation and environment of the Maude language we can run this specification with respect to concrete agreements between different customers and their respective accounts. The first aim is to check for ambiguity and misconception. Then, as second level we have to tackle inconsistency and conflict between different rules. As third aim, and because Maude is endowed with model-checking properties can be verified. We have to do that independently with respect to both functionalities and contextual concerns. Finally, we have to tackle the integration as we informally described and check again the above issues such misconception, conflict and crucial properties at that formal level.

A detailed specification and validation of the functionalities ECA-driven rules we implemented using the Windows Maude Workstation are shown in Appendix-A.

In order to dynamically integrate different functional and context-aware rules, we take benefits of the reflection of rewrite logic and its implementation as so-called strategies in the Maude language. The following Figure depicts an illustration how different rules can be combined.

The semantical framework we are proposing for this new concern-based ECA-compliant service oriented architectural conceptualization is based on rewriting logic [Mes92], which has been proved very appropriate for dealing with concurrent systems. Further strengths making this logic very practical is the current implementation of the MAUDE language [3, 8]. In MAUDE object states in are conceived as terms — precisely as tuples— of the form <Id : C|at1 : v1, .., atk : vki>. In this tuple : Id stands for object identity; C identifies an object class; and at1, ..., atk denote attribute identifiers with v1, ..., vk as current values. Messages (i.e. method invocation) are regarded as operations sent or received by objects, and their generic sort is denoted Msg.

Object and message instances flow together in the so-called configuration, which is a multiset, w.r.t. an associative commutative operator denoted by ' ', of messages and (a set of) objects. The effect of messages on objects is captured by appropriate rewrite rules.

Without delving into detail about how to we systematically allow deriving the rewrite rules from the ECA-driven interactions of both concerns, we sketch here directly the Maude code corresponding to the withdrawal case, which should look like this module.

Then using the current implementation and environment of the Maude language we can run this specification with respect to concrete agreements between different customers and their respective accounts. The first aim is to check for ambiguity and misconception. Then, as second level we have to tackle inconsistency and conflict between different rules. As third aim, and because Maude is endowed with model-checking properties can be verified. We have to do that independently with respect to both functionalities and contextual concerns. Finally, we have to tackle the integration as we informally described and check again the above issues such misconception, conflict and crucial properties at that formal level.

A detailed specification and validation of the functionalities ECA-driven rules have implemented as depicted in the following Figure A-1 (in Appendix-A). We should further point out that the execution of these rules can be dynamically controlled using the so-called strategy as a reflection-based manner to control the order in which different rules can be executed. An illustration of such strategy for the withdrawal Activity is depicted in Figure A-2 in the Appendix.

8. CONCLUSIONS

In this paper, we put forwards a service-oriented architectural-based approach that addresses current challenges in modern business process modelling for reflecting dynamic cross- and intra-organisational interactions as well as context-aware dependencies. We proposed ECA-driven semantics primitives to separately model, evolve and validate both concerns at the activity-level. We further explained how these concerns to-be integrated to reflect the intuitive business semantics of any business activity. Rewriting logic and its Maude language have been proposed for the formal validation and verification of both concerns. Furthermore, in order to dynamically integrate different functional and context-aware rules, we have taken benefits of the reflection of rewrite logic and its implementation so-called strategies in the Maude language.

To further consolidate and validate this service-oriented architectural approach we are working on more case studies. Among the most interesting and practical fields that we are working on is the healthcare, where context-ware beds, specific heart-devices and other context-intensive medical tools are becoming nowadays more that ubiquitous [5, 27]. We are also implementing the different phases of the approach. One of our main goals is to develop a deeper understanding and classification of business rules so that semi-automatic derivation of functionalities and context-aware architectural ECA-driven connectors can be ultimately achieved.

REFERENCES

[1] R.Allen and D.Garlan, "A Formal Basis for Architectural Connectors", ACM TOSEM, 6(3), 1997, 213-249.

[2] N.Aoumeur, J.Fiadeiro and C.Oliveira, "Distribution concerns in service-oriented modelling" Int. J. Internet Protocol Technology, Vol. 1, No. 3, 2006.

[3] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and C.L: Talcott. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. Lecture Notes in Computer Science (springer), 4350, 2007.

[4] P.Cong Vinh, N.Tat Thanh and H.Chi Minh (eds.), "Context-aware Systems and Applications, and Nature of Composition and Communication", 7th EAI International Conference, iccasa 2018 and 4th EAI International Conference, ICTCC 2018 Proceedings, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer

[5] P.Cipresso, s.serino and D.Villani (Eds). "Computing Paradigms for Mental Health.", 9th International Conference, MindCare 2019, Buenos Aires, Argentina, April 23–24, 2019 Proceedings, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2019.

[6] P.C Dockhorn, P.A. Almeida, L.F. Pires, and M. van Sinderen. "Situation Specification and Realization in Rule-Based Context-Aware Applications." In Proc. of the Int. Conference DAIS'07, page 3247. LNCS, Volume 4531, 2007.

[7] C.Dobre and F.Xhafa. "Pervasive Computing Next Generation Platforms for Intelligent Data Collection." Academic Press is an imprint of Elsevier. Elsevier, 2016.

[8] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, C. L. Talcott: Programming and symbolic computation in Maude. J. Log. Algebraic Methods Program. 110, 2020.

[9] A.Juan Fuente, , B. López Pérez, G. Infante Hernández and L.J.Cases Fernández, "Using rules to adapt applications for business models with high evolutionary rates.", International Journal of Artificial Intelligence and Interactive Multimedia, Vol. 2, Nº 2, 2013.

[10] Business Rules Group. Defining Business Rules - What Are They Really? In www.businessrulesgroup.org, 2005.

[11] P.Kardasis and P.Loucopoulos, "Expressing and Organising Business Rules", Information and Software Technology, 2006.

[12] J. Meseguer. Conditional rewriting logic as a unified model for concurrency. Theoretical Computer Science, 96:73–155, 1992.

[13] J.Magee and J.Kramer, "Dynamic Structure in Software Architectures", 4th Symp. on Foundations of Software Engineering, ACM Press 1996, 3-14.

[14] E.Marius Oprea, M.Alexandru Moisescu and S.Caramihai, "Context Awareness in Enterprise Systems Design, May 2021." In 23rd International Conference on Control Systems and Computer Science (CSCS), 2021, DOI: 10.1109/CSCS52396.2021.00053.

[15] L.Mutanu and G.Kotonya, "State of runtime adaptation in service-oriented systems: what, where, when, how and right.", Special Issue: Adaptive and Reconfigurable Service-Oriented, Cloud and Virtualised Architectures, IET Software, Vol. 13 Iss. 1, pp. 14-24.

[16] G.J. Nalepa and S.Bobek "Rule-Based Solution for Context-Aware Reasoning on Mobile Devices.", Computer Science and Information Systems 11(1):171–193, 2013.

[17] B.Orrinsi, J.Yang, and M.Papazoglou, "A Framework for Business Rule Driven Web Service Composition", in Proc. of Conceptual Modeling for Novel Application Domains, LNCS 2814 Springer 2003, 52-64.

[18] J.Oukharijane, I.Ben Said, M.Chaâbane, R.Bouaziz, Rafik and E.Andonoff, Eric.. "A Survey of Self-Adaptive Business Processes", In 32nd International Business Information Management Association Conference (IBIMA).", Seville, Spain, Feb 2019.

[19] M.Papazoglou and D.Georgakopoulos (guest editors), Special Issue on Service-Oriented Computing, Communications of the ACM 46(10), 2003.

[20] D.Rosca and C.Wild, "Towards a Flexible Deployment of Business Rules", Expert Systems with Applications 23:385--394, 2002.

[21] M.P. Papazoglou. Web Service: Principles and Technology. Prentice-Hall, Englewood Cliffs,2007.

[22] RuleML: "The Rule Markup Initiative." http://wiki.ruleml.org /index.php /RuleML_Home. 2021

[23] O.Vasilecas, D.Kalibatiene and D.Lavbič, "Rule- and context-based dynamic business process modelling and simulation", Journal of Systems and Softwarem 122, 2016.

[24] W.Wan-Kadir and P.Loucopoulos, "Relating Evolving Business Rules to Software Design", Journal of Systems Architecture, 2003.

[25] T.Elr, "Service-Oriented Architecture: Analysis and Design for Services and Microservices.", Printice-Hall, 2016.

[26] O.Vasilecas, D.Kalibatiene and D.Lavbič, "Rule- and context-based dynamic business process modelling and simulation", Journal of Systems and Software, Volume 122, December 2016, Pages 1-15.

[27] J.Symonds, " Ubiquitous and Pervasive Computing: Concepts, Methodologies, Tools, and Applications", Copyright © 2010 by IGI Global, 2010.

[28] M.Guo, J.Zhou, F.Tang, and Y.Shen, "Pervasive computing : concepts, technologies and applications", Taylor & Francis Group, LLC, 2017.
[29] D.Lupiana, "Architectural Solutions for Context-Aware Applications: KoDA Prototype", International Journal for Information Security Research (IJISR), Volume 9, Issue 1, March 2019

APPENDIX-A

ACNT_CMP_GNR.maude

```
1.  mod ACNT_CMP is
2.     protecting INT .
3.     inc CMP_GNR .
4.     sorts CRDT DBT CHGL TRS His HisL AcntCf AcntId .
5.     subsorts CRDT DBT TRS < obs_Msg .
6.     subsort  CHGL < loc_Msg .
7.     subsorts His < HisL < loc_Msg .
8.     subsort AcntCf < ConfCMP .
9.     subsort AcntId < CMPid .
10.    op Crd(_, _) : AcntId Int -> CRDT [ctor] .
11.    op Db(_, _) : AcntId Int -> DBT [ctor] .
12.    op ChgL(_, _) : AcntId Int -> CHGL [ctor] .
13.    op Trs(_, _, _) : AcntId AcntId Int -> TRS .
14.    op bal: _ : Int -> obs_Prop [ctor gather (&)] .
15.    op limt: _ : Int -> loc_Prop [ctor gather (&)] .
16.    op [] : -> His .
17.    op [_, _] : Int Nat -> His .
18.    op _, _ : His HisL -> HisL .

19.    vars A A1 : AcntId .
20.    vars B B1 L L1 : Int .
21.    var M : Nat .
22.    rl [credit] : Crd ( A, M ) < A | bal: B > => < A | bal: B + M > .
23.    rl [debit] : Db (A, M) < A | bal: B > => < A | bal: B - M > .
24.    rl [chgl] : ChgL ( A, L1 ) < A | limt: L > => < A | limt: L1 > .
25.    rl [transfer] : Trs(A, A1, M) < A | bal: B > < A1 | bal: B1 >
26.                 => Db (A, M) < A | bal: B > Crd ( A1, M ) < A1 | bal: B1 > .
27.  endm
```

Figure A-1. The Implementation of the Withdrawal Rules using the Windows Maude Workstation

ACNT_STR.maude

```
mod ACNT_STR is
inc ACNT_CONF .
protecting META-LEVEL .
vars withdraw? deposit? transfer? : [Result4Tuple] .
var T : Term .
op Compute : Term -> Term .

ceq Compute(T)
 = (if(deposit? :: Result4Tuple)
   then Compute(getTerm(deposit?))
   else if(transfer? :: Result4Tuple)
        then Compute(getTerm(transfer?))
        else if(withdraw? :: Result4Tuple)
             then Compute(getTerm(withdraw?))
             else T
        fi fi fi)

  if withdraw? := metaXapply(upModule('ACNT_CONF,false),
              T,'withdraw,none,0,unbounded,0)
    /\ deposit? := metaXapply(upModule('ACNT_CONF,false),
              T,'deposit,none,0,unbounded,0)
    /\ transfer? := metaXapply(upModule('ACNT_CONF,false),
              T,'transfer,none,0,unbounded,0) .

eq Compute(T) = T [owise] .
endm
```

Figure A-2. A Strategy-based Implementation of The Withdrawal Rules using the Strategy-Module of the Windows Maude Workstation