Towards a Disciplined Engineering of Adaptive Service-oriented Business Processes

Nasreddine Aoumeur, Kamel Barkaoui, Gunter Saake

Towards a Disciplined Engineering of Adaptive Service-oriented Business Processes

Nasreddine Aoumeur Kamel Barkaoui[1] Gunter Saake
ITI, FIN, Otto-von-Guericke-Universität Magdeburg
Business Information group, D–39016 Magdeburg, Germany
E-mail: {aoumeur | saake}@iti.cs.uni-magdeburg.de
[1]CEDRIC-CNAM, 292 Saint Martin, Paris - FRANCE barkaoui@cnam.fr

ABSTRACT

Today's cross-organizations are increasingly coordinating their capabilities in the quest of dynamically adaptable and thus highly competitive realistic services. Unfortunately, challenging problems are still to circumvent towards such objective, including the inherent rigidity, knowledge-scarce and lack of dependability of most Web-Services standards (e.g. WSDL, BPEL and WS-CDL). We are contributing by putting forwards an integrated model-driven approach, with as main conceptual / deployment milestones and phases the followings. Firstly, at the domain-level, we are bringing profiled UML use-cases and class-diagrams to intuitively capture the structuring of service-driven applications. Secondly, to cope with any required knowledge and its agility, we are governing any business activity with event-driven business rules. Thirdly, towards verified conceptualization, we are shifting these UML-BRules-centric service requirements towards a tailored rule-centric service-oriented Petri nets formalism, we endow with a truly-distributed operational semantics based on rewriting logic. Fourthly, capitalizing on aspect-oriented mechanisms, we progressively upgrade that service formalism with an adaptability aspectual-level, where governing business rules can be dynamically adapted and (un)woven. Finally, towards a compliant deployment, we are developing an aspectual .Net framework for efficiently adapting Web-Services. A typical travel-agency is taken for proof of concepts

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Web-Services; D.2 [**Software**]: Software Engineering, Adaptability

Keywords

Dynamic Adaptability, UML, Business rules, Service-oriented Petri nets, Rewriting logic, Aspectual .Net WS

1. MOTIVATION

The emerging of the service computing paradigm (SOC) is currently establishing a new organizational and business realities. Indeed, SOC is shifting them from traditional centralized computation-centric standing-alone companies to loosely-coupled truly-distributed interaction-centric massive cross-organizations. At the technological level, Web-Services are offering best networking platform-independent infrastructure for externally cooperating cross-organizational business processes [1]. Web-services are the explicit computational units, which can through their interfaces be universally described, published and more importantly (dynamically) composed using XML-based standards (e.g. WSDL, UDDI, BPEL4WS, WS-CDL [13]).

As these standards are maturing, more and more world-wide cross-organizations are opting for service-oriented solutions, and thereby putting at proof all capabilities and limitations while building truly realistic service-driven applications. Adaptability and correctness, besides knowledge-intensivity seem to be the most challenging issues to be addressed towards leveraging these standards towards realistic services [14]. Firstly, whereas WSDL and BPEL are inherently static and manual, in face of the harsh competition and market globalization and volatility, realistic services are deemed to be highly adaptive and evolving. Secondly, whereas most of potential service-driven applications such as E-commerce and E-health and E-banking are becoming mission-critical, BPEL and the others standards are only ad-hocly built without any means to formally validate them. Last but least, whereas most of potential service-driven applications are knowledge-intensive (i.e. geared by business rules [23] and policies), in BPEL only basic variables and primitive conditions can be manipulating in static manner.

We aim thus join the tremendous efforts being invested towards leveraging the service paradigm towards coping with highly agile and knowledge-intensive service-driven applications in stepwise and rigorous model-driven manner. Broadly speaking, the approach we are proposing enjoys the following capabilities, steps and characteristics:

- First, we are proposing to capture initial requirements through UML diagrams [22, 7] and event-driven business rules [27]. More specifically, we propose profiled Use-cases and class-diagrams to informally and diagrammatically express structural features of any (composite) service requirements. Towards tackling adaptability and knowledge-intensiveness, we propose event-driven ECA business rules for governing any business activity taking part in a given composite service-oriented business process.

- To stay compliant with UML-rule centric business description while enhancing it with formal underpinnings, visual validation and high-distribution, we put forward a tailored rule-driven service-oriented high-level Petri nets formalisms. The framework, we refer to as $\mathcal{R}$SRV-NETS is further endowed with a true-concurrent operational semantics in terms of rewriting logic [20] and its efficient declarative MAUDE language [10]. Smooth translating steps are introduced to the bring the business-level to this conceptualization.

WWW2009, April 20-24, 2009, Madrid, Spain.
.

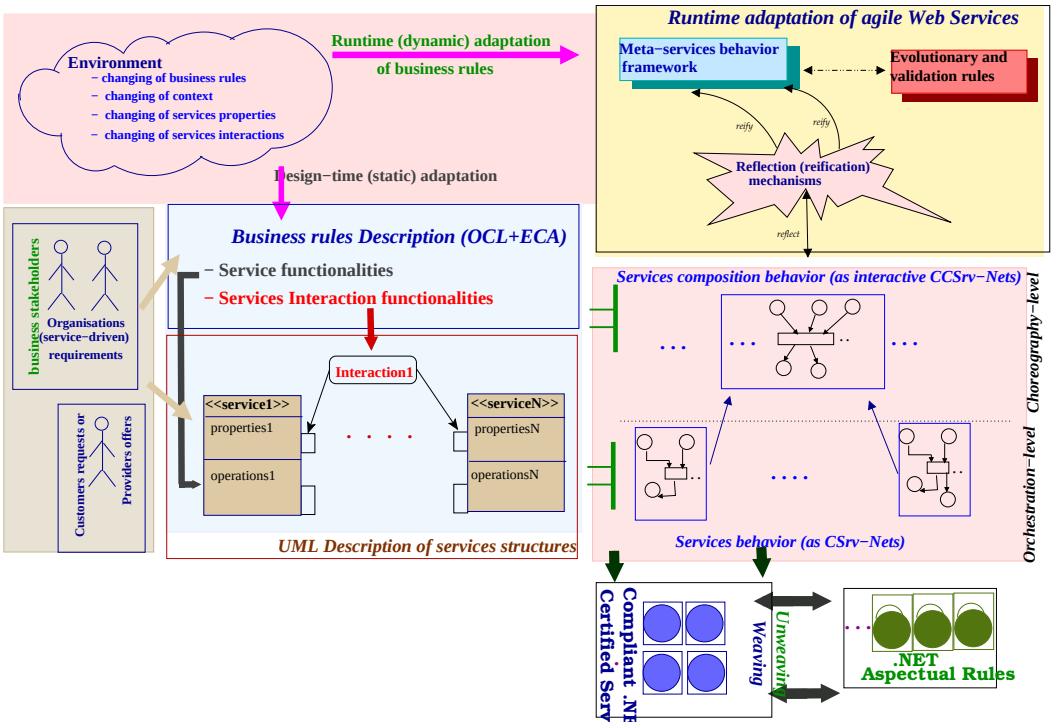

Figure 1: A disciplined approach for adaptive knoweldge-intensive service-driven business applications.

- A further step towards dynamic adaptability is proposed. It consists in recapitulating on aspect-oriented mechanisms [**?**] and meta-reflection [8] to extend that framework to cope with runtime adaptability.
- As ultimate Web-Services-centric deployment we are developing, for this disciplined model-driven approach to adaptive services, a strictly compliant aspectual .NET environment.

The rest of this paper is organized as follows. The second section illustrates and summarizes the working architecture of the approach we are striving for. In the third section, using the travel agency as proof of concept, we detail the semi-formal modelling of services using the profiled UML diagrams and the event-driven business rules. In the third section, we present different steps for formally specifying services with $\mathcal{R}$SRV-NETS, from the previous semi-formal step. The fourth section demonstrates how validation could be achieved using rewriting logic as semantics for $\mathcal{R}$SRV-NETS. In the fifth section, we present how to progressively leverage that conceptual model to cope with adaptability by recapitulating on aspect- and reflection-mechanisms. In the sixth section, we summarize the main features of the aspectual .NET based environment we developing for efficiently implementing the approach using Web-Services technology. In the seventh section we detail some related work on service adaptability. This paper is finally wrapped up by some concluding remarks and further required extensions of this work.

2. THE APPROACH WORKING ARCHITECTURE: PHASES AND MILESTONES

As we motivated, the approach we are proposing for agile and rule-centric service-driven complex applications is stepwise and model-driven. As depicted in the Figure 1, the working general architectural vision of this approach could summarized as being methodologically composed of four phases.

UML/Business-rules Requirements phase : In this preliminary phase, the informal description of the composite service-driven application at-hand is semi-formally and diagrammatically expressed in terms of UML Use-Cases and Class-diagrams. Besides that, all related intra- and inter-organizational business rules governing the behavioral features of different basic and composite business activities are to be clearly described, following in particular the well-known Event-Condition-Action (ECA) paradigm.

Concurrent services Nets specification / validation phase : During this decisive phase, should be precisely defined all functionalities and behaviors of different service components and their interactions (i.e. service interfaces, elementary and composite services). Furthermore, we propose to formally validate them against misconception, conceptual errors, etc. For this crucial phase, we are thus proposing a tailored variant of service-driven high-level Petri nets, that reflects all structural and behavioral features of elementary or composite rule-centric services, such as distribution, persistency (statefull) and conversation and complex structuring mechanisms (e.g. classification, inheritance, aggregation). For validation purpose, we are semantically interpreting this formalism using rewriting logic.

Aspectual service Nets for runtime evolution : For the purpose of dynamic adaptability of the governing rules of any activity, we are extending the above service formalism with an explicit adaptability-level. This adaptability-level is conceived by recapitulating on aspect-oriented mechanisms, where rules are

conceived as advices to be dynamically (un-)woven on respective transitions as business activities.

A Compliant .NET based environment : For the ultimate deployment phase, we are developing a strictly compliant environment that preserves all the capabilities of that conceptual level yet exploit Web-Services technology and aspect-oriented mechanisms at the infrastructural-level

3. UML DIAGRAMS AND BUSINESS RULES FOR SERVICE REQUIREMENTS

3.1 Travel Agency : Informal description

In the simplistic case, a travel agency sells flight tickets and reserves hotel rooms. In order to provide these services for its customers, a travel agency needs to establish business links with other enterprizes, i.e. airlines and hotels. In this context, a financial institution, i.e. a bank, is required to facilitate the financial transactions between a customer and a business or between a business and another business.

In order to make a trip, the customer accesses the Web service of a travel agency that sells flight tickets and provides hotel rooms reservations. The customer enters his requirements. The travel agency receives the requirements of the customer and send them to different airlines and hotels. The travel agency receives the possibilities from these partners and chooses the best solutions for flights and hotels. It sends them to the customer who chooses, reserves and pays for this (holiday) package. The payment is made with the support of a bank usually using a credit card.

3.2 UML diagrams and Business Rules

To illustrate this simplified variant of vacation arrangement, Figure 2 presents its corresponding use-case diagram. This use-case made the relationship between the activities to do and different services with the travel-agency service coordinating these activities.

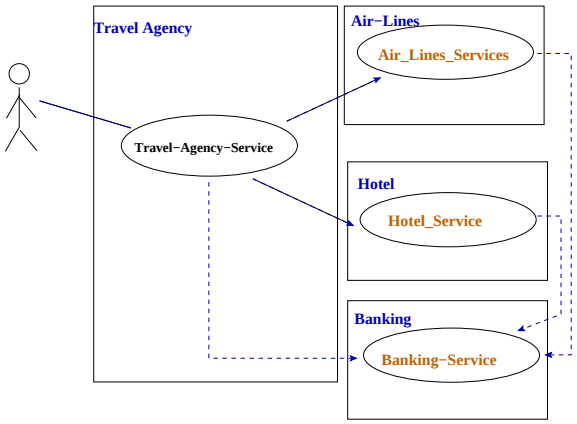

Figure 2: A UML Use-Case for the Travel Agency Case Study.

The next Figure 3 goes in detail about the different classes (as services) and their interactions to the travel-agency. We should point out that detail about message parameters and other properties can be kept semi-undefined as it should precisely defined in the next formal phase, depending on the business rules to put in place.

This case study is one of the most adopted and at the same time the most *volatile* one. To stay competitive and attract more customers, each travel agency has to offer the wider possible range of vacation packages depending of environmental situations (i.e. seasons, events, years, short/long vacation, etc.) as well customers preferences and situations (i.e. individual/group, complete/partial packages, etc).

To specify such service-driven business applications, as any complex reactive distributed system we have to cope with structural as well as behavioural requirements. With the defacto standardization of UML diagrams for structural aspects (i.e. class- and object-diagrams), we argue that UML class-diagrams with slight profiled extensions allow capturing for each service, the operations and properties (attributes) structure. The challenging problem remain the modelling of behavioural aspects, where *reactivity*, *distribution*, *composability* and more especially *evolution* and *adaptivity* has to be the heart of any accepted conceptual model.

At early requirement stages, *business rules* represent the best available modelling ingredients in organisations to cope with competitiveness and evolution. Business rules reflect regulations and conditions for the functioning of any (inter-)organisation internally as well as externally, and thus as regulations change/evolve the rules change [27, 18]. Business rules are mostly expressed in terms of Event-Conditions-Actions (ECA) forms.

The travel-agency functioning has to be governed by business rules, and so each service composing this application (i.e. flight service, hotel service, car rental, attraction service, etc.). Just for illustration, a possible business rule regulating the airlines service could be:

Rule (for flight)*"The fare for a return ticket for a family with childrens is reduced to 30% for each child. When booking before 3 weeks, further discounts of 10% is applied for the adults."*.

With respect to business rules change, we distinguish between adaptivity and evolution. Both are *effects* that are *caused* by changes in the governing rules and environment. For adaptivity, the changes are usually made at *runtime* whereas in evolution changes happen over a long period of time and statically. For instance, we may assume refund system to change according to changes in the environment (e.g. flight delay). In contrast to that, evolution is more concerned with the introduction of new rules or the establishment of new service operations (e.g. mobile-calling on flights)

Besides adaptivity and evolution, distribution remains one of the essential feature of service-orientation computing. It includes in our case the possibility of requesting vacation services from anywhere as well as the possibility of serving simultaneously several requests (i.e. true-concurrency). Related to distribution is the reactivity feature. Reactivity implies stateful modelling, where a given transaction could be long-running (especially when we require history). For instance, a customer should have the opportunity to change at any time some of information and requirements. To cope with that, service states instances have to be explicitly represented in the model. Last but not least, while it easy to conceive at *design-time* new services, it is more beneficial to adapt existing rules at runtime without stopping the system or decreasing its degree of distribution.

For all these considerations (i.e. distribution, reactivity, runtime adaptivity and design-time evolution), the next section presents the approach we are working on that is based on a form of high-level Petri nets. This formal conceptual model will be automatically derived from UML-class diagrams and business rules.

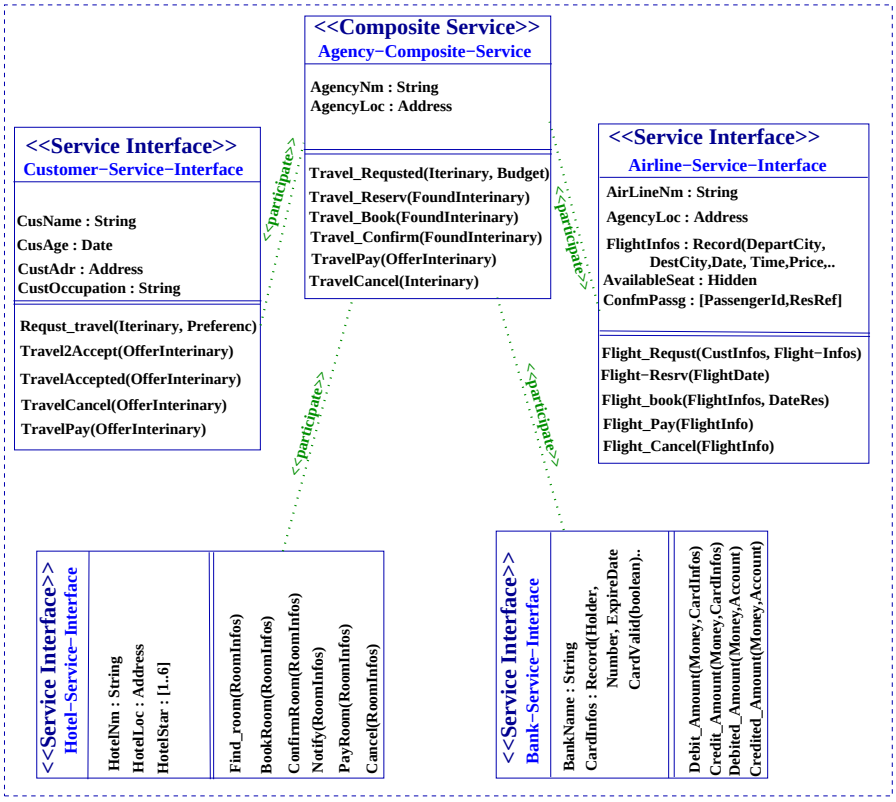

Figure 3: The Travel Agency with a SteroTyped UML Class-diagram for Services.

4. FORMAL MODELLING OF SERVICES STRUCTURE: $\mathcal{R}$SRV-NETS STATES

The first step towards formalizing service-driven applications consists in precisely defining different states and messages accepted by basic service interfaces as well as composite services. In the approach we are proposing, as we already emphasized we endeavor benefiting from advanced structuring mechanisms of the object-orientation (i.e. classification, inheritance, composition and aggregation) and thereby also facilitate the derivation of formal service interface structures from UML class-diagrams and business rules we described in the previous section. So, in our approach besides the description of messages signatures (as most of XML-technology languages offer), the precise description of service *states* enables us afterwards to specify the *statefull* concurrent behaviour of service interfaces: a capability completely missing in XML-based languages (e.g. WDSL, BPEL, etc.) and only partially addressed in recent formalisms to service specification (e.g. Petri Nets [19] Graph-Transformation [17], Temporal Logic [26], Process Algebras[15].

More precisely, we propose to specify service states as algebraic terms in the form of specific *tuples*. These service states as tuples although inspired by the structure of the MAUDE language [21] object states, they enjoin very specific properties reflecting at the most the main characteristics of service interfaces. More precisely, the structure of service states we are following can be informally explained as follows:

- Any service state is conceived as an algebraic term of the form

  $\langle SvId \mid sv\_pr_1 : vl_1, ..., sv\_pr_p : vl_p, sv_{h_1}(SvId), ..., sv_{h_q}(SvId) \rangle$

  where

  – $\texttt{SvId}$ is interpreted as an observed service state identity taking its values from a given appropriate abstract data type ADT (that we assume denoted as $\texttt{STId}$);

  – $\texttt{sv\_pr}_1, \ldots, \texttt{sv\_pr}_k$ are the observed identifiers for service state properties or attributes, which we assume having at a given time as current values respectively $\texttt{vl}_1, \ldots, \texttt{vl}_k$. We assume both service states identifiers and values to be algebraically defined (elsewhere), by denoting their respective ADT as $\texttt{SPId}$ and $\texttt{SP\_Value}$ (as abbreviation for *S*ervice *P*roperties *Id*entifiers and *S*ervice *P*roperties *Value*s).

  – To enhance privacy, we allow *hiding* values of specific service state properties when required. To declare such Hidden service state properties we adopt the notation of "attribute-as-functions"; so if for instance the value of an attribute identifier, denoted by $\texttt{sv}_{h_1}$, is to be hidden, we denote it as a function $\texttt{sv}_{h_1}\texttt{(SvId)}$, with $\texttt{SvId}$ the corresponding service state identity.

- Messages involved in a given service interface are also specified as algebraic operations. Since messages act on service state instances, they should include as parameters at least one state identifier. Moreover, in a given service interface some messages may be declared to act only on states within this interface; other messages may be exported to participate in a composite service interaction (as a choreography) or take part in other service interface description (as an orchestration), and finally messages may be imported from other interfaces to constraint the messages flow in such service interface (as

allowed by BPEL orchestration). In other words, in a given service interface three categories of messages may be distinguished

Local messages : These are messages that are declared and exclusively exchanged within a given service interface. They either act for state changes in such service interface and/or allow participating and controlling the flow (i.e. the business process) of such service interface.

Imported : These messages are declared in other service interfaces and used by the given service interface in the message flow for orchestration purpose (as BPEL proposes for instance).

Exported : These messages are declared within a given service interface and used by other service interfaces or by compose services.

To bring more understandability and expressivity in our service interface formal structure, we thus explicitly distinguish between these three types of messages. This allows us afterwards to address their corresponding specific behaviour adequately. To formally capture this intuitive description of $\mathcal{R}$SRV-NETS service interface structure specification, first we define the notion of ($\mathcal{R}$SRV-NETS-)service state. Importantly to mention that this modelling is heavily inspired by our previous work on component Petri nets for developing information systems [2, 4, 3, 5].

Definition 4.1 (SERVICE**-state template)** *A service state is defined as a pair* $(Sv_D \cup ST_{Sv}, \{Op\}_{ST_{Sv}})$ *with:*

- Sv_D *is a set of (service data) sorts with:* $\{Bool, STId, SPId, SP\_Value\} \subset Sv_D$. *To allow aggregate service states, we allow* $STId$ *to be subsort of* SP_Value *(i.e.* $STId < SP\_Value$*).*
- ST_{Sv} *is a set of service state sorts (different from* Sv_D*), which we assume contains at least one sort (so we can speak about statefull service interface).*
- $\{Op\}_{ST_{Sv}}$ *is a set of service state operations indexed by* $STId \times (SPId \times SP\_Value)^+ \times ST_{Sv}$. *More precisely, with each service state sort from* ST_{Sv} *a service state operation is associated reflecting the corresponding tuple of such service state sort.*

Remark 4.1 *This important concept of service state structure leads to the concept of* $\mathcal{R}$SRV-NETS *template specification by extending it with involved service message sorts and service operations.*

Definition 4.2 (SERVICE**-template specification)** *is defined as a pair* $(Sv_D \cup ST_{Sv} \cup Msg_{Sv}, \{Op\}_{ST_{Sv}} \cup \{Op\}_{Msg_{Sv}})$ *with:*

- $(Sv_D \cup ST_{Sv}, \{Op\}_{ST_{Sv}})$ *is a service state structure as defined above.*
- Msg_{Sv} *is a set of 'message generator' sorts different from* $Sv_D \cup ST_{Sv}$. *We assume that* Msg_{Sv} *is composed of three sets of message sorts:* $\{Mes_{l_1}, ..., Mes_{l_l}\}$ *for local message sorts,* $\{Mes_{i_1}, ..., Mes_{i_i}\}$ *for imported ones and* $\{Mes_{e_1}, .., Mes_{e_e}\}$ *for exported ones.*
- *The message operations,* $\{Op\}_{Msg_{Sv}}$ *is a set of message operations, that is, operations indexed by* $STId^+ \times Sv_D^* \times Msg_{Sv}$. *Thus, with each message sort* Mes_{ij} *from* Msg_{Sv} *a message operation (denoted* ms_{ij}*) is associated.*

4.1 Application to the Airlines Service

Following this $\mathcal{R}$SRV-NETS- service template, the corresponding service structure corresponding to the flight service interface, for instance, can be straightforwardly derived from the semi-formal UML service-diagram description we presented in section 2. More precisely this description is as below, where first we have to specify all imported abstract data types allowing to specify different properties and parameters of service states and their messages. These datatypes should include, for instance, the city of departure and of destination (we abbreviate by `Dest` and `Depart` both of sort string). Reservation and confirmation codes have to be specified (abbreviated by `RsvRef` and `CfrmRef`). Date of departure and of return, the maximal cost not to go beyond as well as the fare of flight have to be declared. Information about any customers (package), such as names, addresses, ages, number of adults, child and infants we gathered in one sort denoted $CUST\_INFO$. Such information and more similar are crucial for expressing different business rules later in the service net behaviour. Additionally, to keep track of different passenger reservations and bookings we have previewed a list composed of the customer ID and an identifier indicating whether its reservation or booking (e.g. "Rsv" for reservation and "Bk" for booking).

```
Service Flight-Service is
  extending Service-state
  protecting AirLine-Data.
  subsort FlghtId AirLId< STId .
  subsort Flght_St < Srv_State
  subsort CHK_SEAT < local_Msg.
  subsort FLGHT_RQ FLGHT_RSV FLGHT_BK
       FLGHT_CL < imported_Msg.
  subsort FLGHT_RQSTD FLGHT_BKD FLGHT_CLD
       PAY_FLGHT PAY_PNLY < exported_Msg.
  (* AirLine State Properties *)
    op ⟨_ | AirLId : _, FlInf : _, AvSt(FlighId),
      RsvP : _, CmfP : _, DlRs : _⟩ :
        FlghtId string FLG_INFOS nat PSSGS
          PSSGS Date→ AirLine_State.
 /* Local messages */
    op ChkSt : FlgId Bool → CHK_SEAT .

 /* Imported i.e. received messages */
    op FlgRq : CustId CUST_INFO AGCYId AirLId
       RQFLG_INFO → FLGHT_RQ .
    op FlgRs : CustId CUST_INFO AGCYId AirLId
       RQFLG_INFO → FLGHT_RSV .
    op FlgBk : CustId RsvRef CUST_INFO
      BK_INFO → FLGHT_BK .
    op FlgCl : CustId AGCYId ClRef → FLGHT_CL .

 /* Exported i.e. invoked messages */
    op FlgRqsd : CustId FLG_INFO AGCYId
      AirLId RsvRef → FLGHT_RQSTD .
    op FlgBkd : CustId AGCYId AirLId
       BkRef → FLGHT_BKD .
    op FlgCld : CustId AGCYId AirLId
       BkRef → FLGHT_CLD .
    op Payflg : CustId AGCYId BkRef
       money → PAY_FLGHT .
    op PayPnlt : CustId AGCYId BkRef
       money → PAY_PNLTY .

   (* Variables to use in the service net *)
   vars Dy : Date ; Gc:AGCYId .
endo .
```

Using this flight data level specification as well as the general service state description and being bounded to the general form of service structure, the corresponding service structure of the flight interface could be presented as follows. That is, first for referring service state flight sort we introduce a state sort we are denoting by `Flight_St`. Instances of flights are identified by the sort `FlghtId`. Secondly, for each message declared in the corresponding UML class-diagram, we associate a corresponding sort and an operation. For instance, for the message `FlightRequest` we declared the sort `FLGHT_RQ` and an (imported) operation we abbreviate by `FlgRq`, and which should have parameters like information about the customer, the agency ID, and clearly all detail about his/her flight itereniary and preferences. The same reasoning is to be applied to all other messages. The service state is constructed by gathering in a tuple-like all properties of the flight from the UML class-diagram specification. The flight state is identified by the FlightId and is composed of the Airline name, all information about the flight (e.g. departureCity, DestinationCity, DepDate, DepTime, ArrDate, ArrTime), the number of still available seats (Denoted by `AvSeat(FlghtId)`), the list of customers (IDs) reserved or booked.

5. BEHAVIOURAL MODELLING OF SERVICES: $\mathcal{R}$SRV-NETS TRANSITIONS

In the previous section we presented how given a service structure specification denoted by $TSrv$ captures the structural aspects of a given service interface. In this section we address the behavioral concerns by incrementally constructing it from the service structure specification and the business rules. We refer to such behaviour issues as CSRV-NET as the behaviour is a form of high-level Petri nets tailored to this service modelling and service structure specification. A service specification as a whole is hence a pair composed of $Serv_S P =\prec TSrv$,CSRV-NET$\succ$.

5.1 Service Net structural Features

Informally speaking, the net to be associated with a given service structure specification is constructed as follows.

- The places of the net are precisely defined by associating with each service message generator one 'message' place.
- With each service state sort a 'state' place is associated.
- Transitions, which may include conditions, reflect the effect of messages on service states to which they are addressed.

5.2 The flight $\mathcal{R}$SRV-NETS service

With respect to the above flight service structure specification, the application of these behavioral constructions result in the following flight CSRV-NET behavioral interface model as depicted in Figure 4.

As defined above, for each service state and message sort a corresponding (typed) place are conceived. That is, to the service state sort `Flght` corresponds a service state place we denote by `Flight-St`. This service place regroups thus all flight state instances in accordance with the flight service structure specification. On the other side, with each service message a corresponding message place is constructed. So, for the three received (i.e. imported) messages (from the agency composite service as will be detailed later) namely `Flight-Request`, `Flight-Booking`, `Flight-Cancel` correspond three associated sort *messages* places. Also, for the five invoked (exported) messages places, namely `Flight-Requestd`, `Flight-Booked`, `Flight-Canceld`, `PayFlight` and `PayPenalty` correspond five messages. Besides that, in order to capture all different exceptions and errors related to different behaviour, we have added another message place we denote by `FlghtOP-Err`. As we will explained subsequently this place receive all attempts for violating the business rules related to different message functionalities.

5.3 Service Net Behaviour Using Business rules

The crucial contribution and added-value of our approach to the service paradigm concerns thus the concurrent behaviour that we are able to assign to different messages and services states. Such behavior will be clearly captured by different *transitions*, with their inherent inscriptions and conditions. For that purpose, that is, for the conception of different transitions, we mainly relies on the *business rules* governing at this stage the (intra-)organization at hand, which the is flight company in our case.

More precisely, first we propose to follow the widely dominating forms of business rules, that is, the *Event-Conditions-Actions* pattern. With respect to the formal definition of $\mathcal{R}$SRV-NETS, a transition in its general pattern allows interacting some triggering messages with service states, leading to the change of invoked states, absorption of the triggering messages and apparition of new invoked messages; all this reaction is of course to be allowed under valid conditions.

For this similarity, it becomes very straightforward how a given ECA business rules can be translated into a $\mathcal{R}$SRV-NETS transition that correctly reflects its behaviour. That is:

event-part : It corresponds to different input messages inscriptions involved in this event part of the rule.

condition-part : it has to be translated into a compatible transition condition. Information related to service states have to translated into input arcs inscriptions from the service state place

action-part : It to expressed in terms of exported messages and changes in the involved service states.

For the flight service interface, we have associated three transitions (i.e.) to reflect the (business) semantics of the three received messages, namely: The transition `Tflgh_rq` for capturing the request activity with the offered flights (i.e. flight_requested) as output result; the transition `Tbkfl` for capturing the booking activity and finally the transition `Tclfl` to gouvern the cancel activity if any.

In the following, we detail the rigorous inscriptions of each of these transitions with respect to very simple business rules (BR). Afterwards, we hint how any complex business rule governing these transitions can be straightforwardly reflected into formal inscriptions in the $\mathcal{R}$SRV-NETS formalism.

5.3.1 The flight $\mathcal{R}$SRV-NETS service behaviour

A typical business rule governing the behavior for flight-request activity in this flight service could formulated as follows.

Rule (for flight)*"On a customer request for a flight, the offered flights have to* ***exactly*** *match the departure and destination cities, the dates and the time of the customer wish. The flight cost should not exceed what the customer tolerates and finally if the customer is minor (less than eighteen years) a reduction of 2% is granted"*.

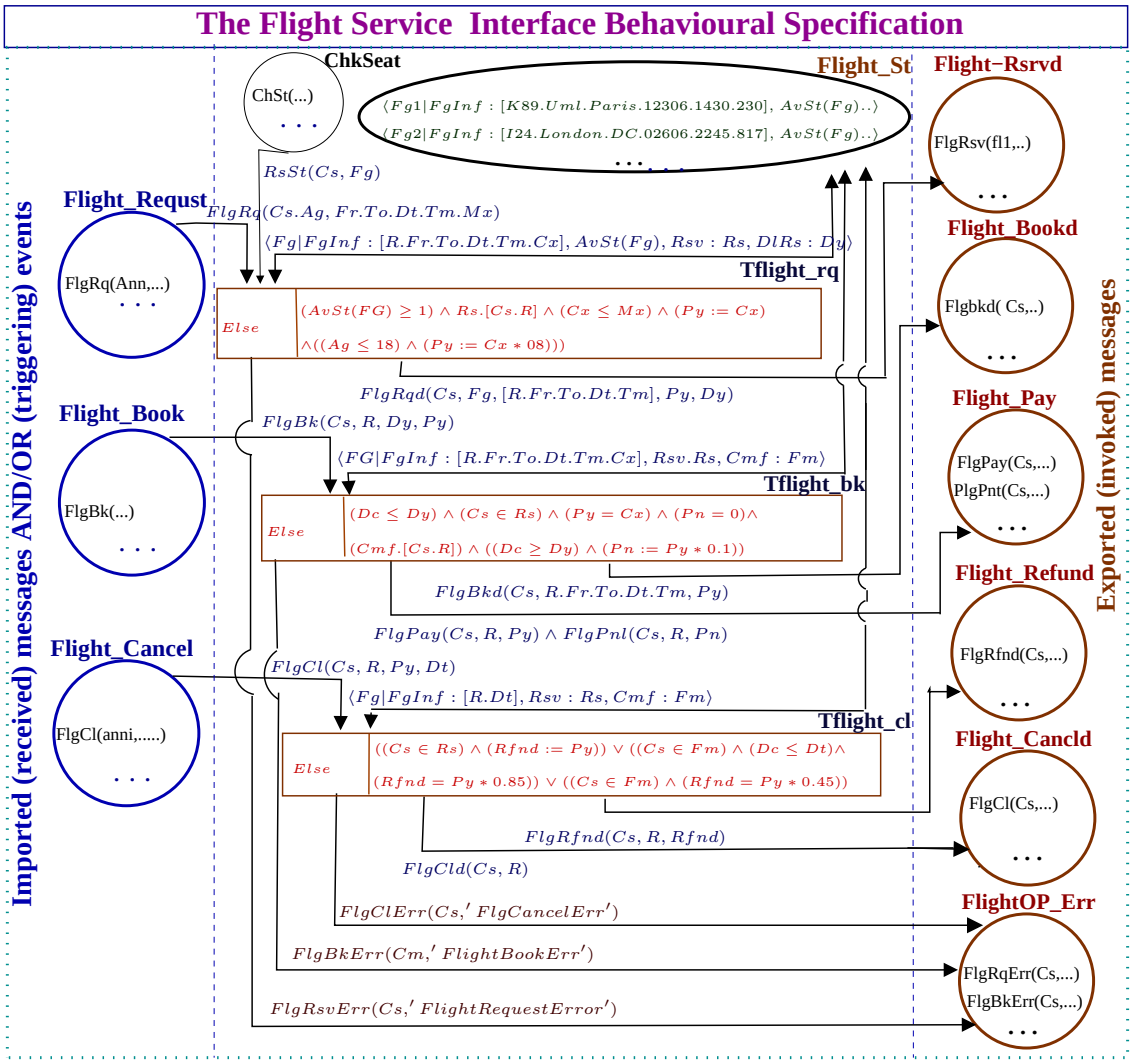

Figure 4: The Behavioural Specification of the flight Service

Following the above guidelines, from this informal rule description the construction of the corresponding transition (`Tflight_rq`) inscriptions could be summarized as follows:

1. To reflect the events part of this business rule, the transition `Tflgh_rq` must have one input inscription from the request message place `Flight_Requst` and one from the state flight place `Flight_St`. By respecting the template message structure and declared variables, the inscription of the request message place takes the form:
   $FlgRq(Cs.Ag, Fr.To.Dt.Tm.Mx)$, that is, the parameters are "CusName, Age, From city, To city, Date, Time flight and max cost to bear. The selected (abstract) flight should have the same information (i.e. same variables). That is, the inscription from the flight state place could be: $\langle Fg|FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg), Rsv : Rs, RsD : Dy\rangle$, with R as the flight reference, Cx as the (normal) ticket price, and Dy the date limit for booking and lose the reservation.

2. To reflect the rule conditions, the transition condition should have the form: $AvSt(Fg) - 1 \geq 0 \wedge Rs.[Cs.R] \wedge ((Cx \leq Mx) \vee ((Ag \leq 18) \wedge (Py = Cx * 08)))$. That is, the available seats has to be decreased by one and be still positive; The reserved list has to be updated to include the new customer and the flight reference, the ticket price Cx has to be less than customer max, and finally if the age is less than 18, the payed amount will be just 80% percent of the price.

3. Finally, the output message to report back the founded reservation with the flight references, the computed price and the date limit.
   This is reflected by the inscription: $FlgRqd(Cs, Fg, R, Py, Dy)$

In the same spirit, we can formulate any business rules for the reservation as well as the cancelation activities and construct consequently the associated net transitions (i.e. `Tbkfl` and `Tclfl`). The reservation business rule should reflect the respect of the reservation deadline (ie reservation information should be still in the corresponding flight state); otherwise a penalty is to be paid beside the ticket price. The corresponding transition is easily conceived as depicted. The cancelation after a booking incur the payment of a penalty.

6. REWRITING-LOGIC SEMANTICS FOR $\mathcal{R}$SRV-NETS

Besides ordinary graphical animation, we endow the $\mathcal{R}$SRV-NETS approach with a *true-concurrent* operational semantics expressed in rewriting logic [20]. That is, each transition is governed by a corresponding rewrite rule interpreted in a tailored instantiation of this logic, we refer to as $\mathcal{R}$SRV-NETS rewrite theory. The main ideas of this interpretation can be sketched as follows:

- To bind each place marking mt with its corresponding place p we capture them as a pair (p, mt). Different tokens within mt are gathered using a multiset union operator we denote by $\oplus$.

- To represent $\mathcal{R}$SRV-NETS states as multisets over different the pairs (p_i, mt_i), we introduce another multiset generated by a union operator we denote by $\otimes$. That is, a $\mathcal{R}$SRV-NETS state is described as a multiset of the form: $(p_1, mt_1) \otimes (p_1, mt_2) \otimes \ldots$;

- To exhibit a maximum of concurrency, we allow distributing $\otimes$ over $\oplus$. That is, if mt_1 and mt_2 are two marking parts in a given place p as $(p, mt_1 \oplus mt_2)$, then we can always split it to $(p, mt_1) \otimes (p, mt_2)$.

- To promote maximum of concurrency within a same service state, we propose a deduction rule that *splits / recombines* the service state at need and taking the form.

$$\langle SvI \mid Sps_1, Sps_2 \rangle = \langle SvI \mid Sps_1 \rangle \oplus \langle SvI \mid Sps_2 \rangle$$

  with Sps_i as an abbreviation of "$spr_i : vl_i, .., sv_{h_n}(SvI)$"

Example 6.1 *By applying these general guidelines for deriving transition rewrite rules to the Customer $\mathcal{R}$*SRV-NETS *service interface Net, we result in a rewriting logic-driven operational interpretation, where rapid-prototypes are directly generated using the* MAUDE *environment [11]. Due space limitation, we restrict ourselves just to depicting the transition rule associated with* `Tflg_rq1`.

Tflg_rq:$(WCsRq, WCsRq(Cs, Am, Ad, Ac(Cs), Bk)) \otimes (WCsST, \sim \langle Cs| \rangle)$
$\Longrightarrow (WCsST, \langle Cs|CsN : Nm, Adr : Ad, Bth : Bd, RqLs : [Am.Now]\rangle) \otimes (WCsRqOk, WCsRqOK(Cs, Am, Ad, Ac(Cs), Bk))$
if $(Ag \geq 21) \wedge (Am \leq Max) \wedge (Ad \in List(Addss))$

7. DYNAMIC ADAPTABILITY: ASPECTUAL EXTENSION OF $\mathcal{R}$SRV-NETS

After the stepwise specification and validation, the crucial further phase is how to make such $\mathcal{R}$SRV-NETS interacting components dynamically evolving. In the following we summarize the different steps that permit building on top of each $\mathcal{R}$SRV-NETS component / interaction an *aspectual* layer for coping with runtime adaptivity.

Meta-tokens as transition behavior: As any $\mathcal{R}$SRV-NETS transition is composed of an label, input/output arc inscriptions with corresponding input/output places, and a condition inscription, we first propose to *gather* them as a tuple :

⟨`trans_id: version | in-inscript., out-inscript., cond.`⟩

Where `version` as natural number captures *different* behavior 'versions'. With respect to the inter-component transition general pattern depicted in Figure **??**, this tuple takes a more precise form:

$$\langle t : i \mid (obj, IC_{obj}) \overset{i_p}{\underset{p=i_1}{\otimes}} (Mes_p, IC_p), (obj, CT_{obj}) \overset{h_r}{\underset{q=h_1}{\otimes}} (Mes_q, IC_q), TC(t) \rangle$$

Aspectual-level for Meta-tokens: To allow manipulating—namely modifying, adding and/or deleting—such tuples (i.e. transitions' behavior), we propose an appealing Petri-net-based proposal that consists in: (1) gathering such tuples into a corresponding place that we refer to as a *meta-place*; (2) associating with this meta-place three main message operations—namely addition of new behavior, modification of an existing behavior, and deletion of a given behavior; and (3) as for usual $\mathcal{R}$SRV-NETS components conceive for each of these three message types three places and three respective `meta-transitions` for effectively and concurrently adapting any meta-transition as tuple.

Relating the two levels with read-arcs: Once building such aspectual-level, to dynamically manipulating any transition dynamics the next important steps are twofold. First, we slightly enrich (selected) $\mathcal{R}$SRV-NETS component transitions by just *adding* (meta-) variables (we denote by $IC\_, CT\_, TC\_$) using a disjunction operator (e.g $\vee$) to each of their input/output and condition inscriptions. In term of

aspect-oriented concepts, these variables play the *jointpoints* for dynamically capturing the advices [?]. Transitions with these (meta-)variables are referred to as *evolving* ones. Secondly, in order to permit *weaving* any new behavior (as meta-token) on the component-level "hooked" transitions, we propose to *relate* through *read-arcs* the meta-place with such respective non-instantiated transition.

Weaving meta-tokens as usual transitions: The dynamic weaving consists in *selecting* from the meta-place a given meta-token as *advice* and *transforming* it to a usual (instantiated) transition rule. Given such a non-instantiated meta-rewrite rule, we can then *dynamically select* any particular tuple-as-behavior from the meta-place and derive a usual transition rule. This process is captured by the following inference rule.

With the existence of the following substitutions:

$\exists \sigma_i \in [T_{s(p_i)}]_{\oplus}, .., \exists \sigma_j \in [T_{s(q_j)}]_{\oplus}, \exists \sigma \in [T_{bool}]$

The following usual rewrite rule as the new (k*th* behavior for the transition $t(-)$ is obtained.

$$\frac{\langle t{:}k \ |[\overset{k}{\underset{i=1}{\otimes}}(p_i,\sigma_i(IC_i))]|, |[\overset{l}{\underset{j=1}{\otimes}}(q_j,\sigma_j(CT_i))]|, \sigma(TC_i)\rangle \in M(P_{meta})}{t^{ins}(k){:}|[\overset{k}{\underset{i=1}{\otimes}}(p_i,\sigma_i(IC_i))]| \Rightarrow [\overset{l}{\underset{j=1}{\otimes}}(q_j,\sigma_j(CT_i))]| \quad \texttt{if} \quad \sigma(TC_i)}$$

7.1 Illustration on the Airline conceptual Model

As we pointed out the first step towards endowing any knowledge service-driven CSRV-NETS specification consists in preparing that specification to become adaptable-aware. More precisely, for each of the CSRV-NETS Airline transition, we have to slightly enrich its (input/output) arc-inscription as well as the condition part with a (meta-)variable using the operator $\bowtie$.

The resulting of applying this enrichment is depicted in the low-level of Figure 5. First note that to ease the manipulation, instead of the long name for places (and transitions) we are shortening them. So, for instance, instead of the place name `Flight_Book`, we are using just `FlBk`. Second, because we want that all activities of the flight service to be adaptable, we are enriching all the three transitions; again here for sake of simplicity we are dropping the `Else` part (i.e. the exception cases) in all these transitions. For instance, for the transition ....

7.2 Building and Illustrating the CSRV-NETS Airline Adaptability-level

Towards effectively bringing this runtime knowledge-centric adaptability conceptual machinery to the above Airline CSRV-NETS prepared specification, let us first consider three simplified business rules scenarios: Two (2) concerning the booking request business activity (i.e. the transition `Tflg_rq`) and one new business rule dealing the cancelling activity (ie. the transition `Tflg_cl`). Of course as will understood from this progressive illustration, we able to dynamically deal with any business rule and with respect to any business activity as transition.

These three business rules could informally described as follows:

Flight to a specific destination and period (`R1`) : This rule says, for instance, that any person travelling to Cairo or Istanbul between June and August gets a discount of 50 percent of the normal fare.

Flight to a specific destination for a group (`R2`) : Any two persons travelling for instance to Las Vigas during the month of january will automatically get 30 percent discount.

Seasonial Flight-cancel (`R3`) : This rule stipulates that during the winter (Christmas time) season a refund will be correspond to the VAT, whereas during the summer the refund concerns only the half of the paid price.

The next step after this informal description of any business rule to be dynamically integrated in the running Airline CSRV-NETS specification consists in *formally* expressing it as a five-element tuple with respect to the transition governing the associated business activity. For illustration, we detail how to translate the second informal rule to its precise five-element tuples description.

The business rule `R2` as meta-token. This rule concerns the flight request and thus the transition `Tflg_rq`. This systematically means that it is the second version or alternative besides the default behavior, and thus the counter for the version is to be set to two (2). In contrast to the first rule, this rule is to be triggered by the simultaneous occurrence of two requests (i.e. from two persons). This implies that from the input place `FlRq` (flight-request) we require two messages, one for instance from `Cs1` and the second from `Cs2` but the same other parameters (i.e. origin, destination, date, cost). The third element in the tuple, that is, the output places and their corresponding inscriptions remain unchanged as in the default; so we abbreviate them using the symbol "`-`". The last element in the tuple concerns the condition, which formulated as for the above rule. We should just note that since two persons are at stake, the available seat should greater than 2 and both the two customer Ids (i.e. `Cs1` and `Cs2`) have to be added to the reservation list `Rs`. The flight cost should of course be less than the max budget of any of the two customers. All in all this tuple takes the following form:

$\langle Tflg\_rq : 2 \ | (FlRq, FlgRq(Cs1.Ag, Fr.To.Dt.Tm.Mx))$
$FlgRq(Cs2.Ag, Fr.To.Dt.Tm.Mx)$
$(FlSt, \langle Fg | FgInf : [R.Fr.To.Dt.Tm.Cx], AvSt(Fg),$
$Rsv : Rs, DlRs : Dy \rangle)$, $-$, $(AvSt(FG) \geq 2) \wedge$
$Rs.[Cs1.Cs2.R] \wedge (Cx \leq Mx) \wedge (Fr = \texttt{"Las Vigas"})$
$\wedge (\texttt{"Jan."} = Dt) \wedge (Py := 2 * Cx * .70) \rangle$

As depicted in Figure 5, for simplicity we have skipped all the places and associated transitions for manipulating the rules-as-tuples. In other words, we just assume that the three above tuples have been introduced using the (meta-)transition T_{AdBh} (for the second rule T_{AdBh1} `as it is the second version`. The firing of the (meta-)transition T_{AdBh} three times successively results in the emerging of three tuples in the meta-place `Brs.Airline-Place` as depicted in the upper-layer of Figure 5. That is to say, three new rules have dynamically introduced at that adaptability-level. To keep the Figure manageable we have indeed skipped this self-explained firing. Giving these rules, it is important to emphasize that we can in the same spirit change and/or delete them through the two other (meta-) transitions (and meta-places).

8. COMPLIANT ASPECTUAL .NET SERVICES

As we already emphasized, current Web-Services standards such as WSDL and BPEL are purely process-centric, static and manual. Consequently, directly adopting them for implementing the proposed approach simply means loosing all the strengths we were striving for, namely adaptability, knowledge-intensiveness and separation of concerns among others.

To cope with these limitations, we are instead proposing the .Net [6] environment and its recent extensions with aspect techniques. The .NET framework [28] is Microsoft proprietary programming

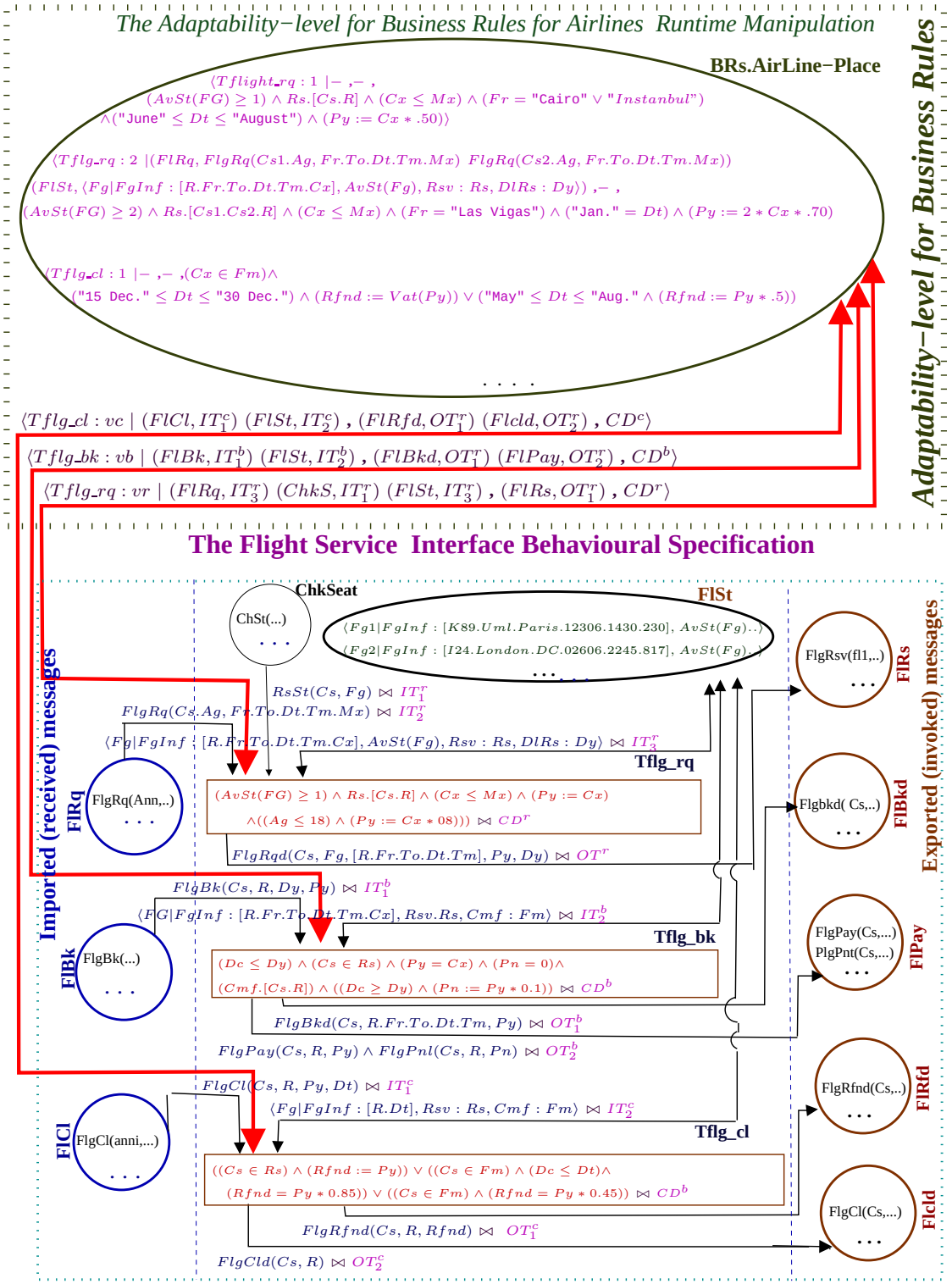

Figure 5: The Runtime Adaptability of CSRV-NETS flight service

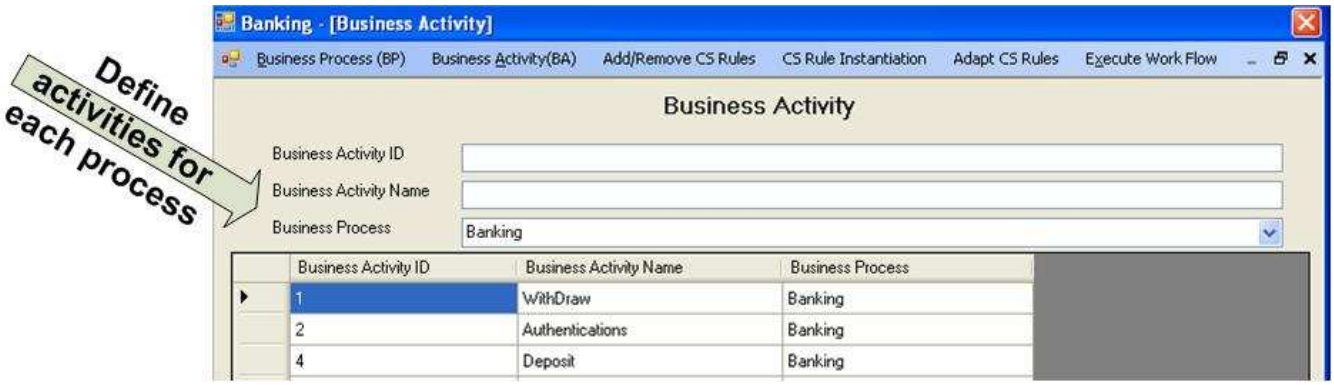

Figure 6: The main functionalities and IDE of the .NET Env.

development environment. It supports syntax for multiple programming languages including C#, VB.NET, J# and C++ all generating Common Intermediate Language (CIL) on compilation. The .NET platform provides excellent support for software extensibility, adaptability, maintainability and customizability through various techniques including reflection, proxy, intermediate language and on-fly code generation.

By fully exploiting these .NET capabilities we are proposing a service-oriented implementation of knowledge-intensive agile business process. This service-based implementation is inherently compliant with the business-conceptual approach milestones, we presented and illustrated in the previous section. More precisely, the main IDE of the .NET supporting tool we implemented inherently reflects the generic pattern definition of service-oriented business process as discussed and depicted in Figure **??**.

As depicted in Figure 6, in this main IDE exposing the implemented environment, first under the icon "`business process`", reference and / or name for any business process can be introduced, deleted or updated. The second icon "`business activity`" allows for manipulating the business activities composing a given business process. As we motivated above, the partial-ordering of such activities within a given business process are postponed to late stages. For instance, depending on the business rules in place, governing a given business process, which are taken in charge thorough the third, fourth and fifth icons, different semantically-driven profiled ordering can be put into place. More precisely, the third icon "Add/remove CS rule" allows for editing, creating or deleting any rewriting (business) rule with respect to given business activity.

Any of the implemented rules can be instantiated as first-class independent entities using the fourth icon "CS Rule Instantiation". As we are striving for dynamically changing any existing business rule, we implemented a main functionality for doing so through the icon "Adapt CS rule". Finally, the last functionality allows for defining a concrete workflow (from the general business process) and executing any case.

After this global presentation of the environment functionalities through its IDE, in the following we first bring more detail on how the event-driven business rules are defined, (design-time) updated, instantiated and then performed. Afterwards, we present how aspect-oriented mechanisms are exploited for *dynamically* adapting and extending any running business rule. Finally, the general architectural design of the software environment is highlighted.

8.1 Mapping of $\mathcal{R}$SRV-NETS Formalism in .NET

To facilitate a smooth yet conservative mapping of the conceptual-level to the .NET service-based implementation, we have been benefiting from the extensive capabilities of the using Microsoft Work Flow Foundation (WFF) [6, 28]. Indeed, WFF supports, among others, the manipulation of (event-driven) business rule using a suitable XML-based templates. It further allows for defining, managing and executing stateful workflow. WFF consists thus of an activity model, workflow designer and XML-based rules engine. RuleDefinitions tag is composed of multiple RuleSets. Each RuleSet is associated to a given business activity. A RuleSet is a collection of rules, where each RuleSet is composed of Rule. Rule contains the (Event-Conditions-Actions) specification as Then-Actions and thus very close to $\mathcal{R}$SRV-NETS rewriting rules.

Figure [**?**] recapitulates the main translating steps of the $\mathcal{R}$SRV-NETS event-driven rewriting rules from conceptual level towards the developed .NET environment. Firstly, as it should be expected different participating service states are internally implemented as Web-Services. In this sense all exposed functionalities for a given rule-driven interaction are to be efficiently implemented with service components. The exposed functionalities, we require to achieve any given interaction are of course captured as Web-Service interfaces (described using WSDL). The second and main translation concerns the mapping of the ECA-driven rewriting rules themselves. First, we capture the rule itself as a (composite) Web-Service, which can be mostly owned by any of the involved providers; though we also consider the case of third-party ownership. For instance, in our application the rules are normally owned by the bank; but they can be outsourced to a third-party for more optimal, universal and intelligent management. Second, the ECA-rule for a given business activity are conceived as a workflow. In this ECA-driven intra-activity workflow, instantiations of the rules can be performed. Since, each rule is implemented as an aspect, the three elements (event-condition-action) composing a rule can be woven on the basis of the instantiated rules.

9. RELATED WORK

The potentials of business rules in Web-services have firstly been explored in [23]. A stepwise rule-driven methodology is proposed to enhance the dynamic adaptability while composing Web-Services using BPEL-like standards. The milestone consists in judiciously classifying in a repository different business rules specific to Web

Figure 7: Rule-centric Instantiated Services execution

$\mathcal{R}$SRV-NETS **Conceptual Level**	**.NET mechanisms**
$\mathcal{R}$SRV-NETS Service states	Service components
$\mathcal{R}$SRV-NETS states structures	WSDL-based interfaces
$\mathcal{R}$SRV-NETS rewriting rules	WFF-based workflow

Figure 8: Translating steps from $\mathcal{R}$SRV-NETS semantics to the compliant .NET env.

Services composition. Nevertheless, no formal verification / validation of the constructed composition is undertaken. Another proposal [24] conceives business rules as Web-Services described using extensions to reactive RuleML, instead of the passive and static WSDL. The approach is automated with supporting tool called ViDre. Nonetheless, the approach does not tackle the conceptualization level neither copes with dynamic composition of the (WS) rules to specific business process *activities*.

A. Charfi et al. [9] leverages BPEL-like with more agility and modularity, by enhancing it with an extra aspectual level. The resulting new language named AO4BPEL, allows thus externalizing cross-cutting concerns such as security and data handling as advices. AO4BPEL uses XML-Query language `XPath` as its pointcuts language. The approach to agile services introduced by Erradi et al. [16] also adopts aspect-orientation, and is specifically devoted to policies and QoS concerns. In the same line, A.Finkelstein et al. proposed in [12] a generic aspect-oriented language, they applied for dynamically weaving behavioral advices on BPEL code. Another aspect-driven approach to Web-services appeared most recently in [25]. In this approach the emphasis is on the adaptability of business *protocols* while composing Web-services.

10. CONCLUSIONS

In this contribution we put forwards a model-driven stepwise approach for describing, specifying, validating, adapting and efficiently deploying distributed rule-centric service-driven applications. More precisely, the first phase consists in describing (semi-formally) structural and functional requirements using UML class-diagrams and business rules in a form of ECA statements. The second phase allows deriving from the first phase a more rigorous specification using tailored High-level Petri nets for services. Validation using graphical animation are possible, we support with formal validation using true-concurrent rewriting logic. The third phases, recapitulates on aspect mechanisms and reflection and propose to endow any service Net with a rule-centric aspectual-level, where rules are dynamically (un)woven on running $\mathcal{R}$SRV-NETS services. Finally, we proposed a compliant aspectual .Net environment for supporting the approach at the implementation-level using Web-Services technology.

For more consolidation we are carrying several non-trivial case studies including in particular a realistic variant of an E-commerce case study. Second, we are developing extensive tools supporting, we are integrating the MAUDE language for rapid-prototyping and formal verification purposes.

11. REFERENCES

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer-Verlag, 2004.

[2] N. Aoumeur and G. Saake. A Component-Based Petri Net Model for Specifying and Validating Cooperative Information Systems. *Data and Knowledge Engineering*, 42(2):143–187, August 2002.

[3] N. Aoumeur and G. Saake. Integrating and Rapid-prototyping UML Structural and Behavioural Diagrams Using Rewriting Logic. In A.B. Pidduck, J. Mylopoulos, C.C. Woo, and M.T. Ozsu, editors, *Proc. of the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02)*, volume 2348 of *Lecture Notes in Computer Science*, pages 296–310, 2002.

[4] N. Aoumeur and G. Saake. Dynamically Evolving Concurrent

Information Systems: A Component-Based Petri Net Proposal. *Data and Knowledge Engineering*, 50(2):117–173, 2004.

[5] N. Aoumeur, G. Saake, and K. Barkaoui. Incremental Specification Validation and Runtime Adaptivity of Distributed Component Information systems. In *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, pages 123–136. IEEE Computer Society, 2007.

[6] T. Archer and A. Whitechapel. *Inside Microsoft C#*. Microsoft Press, 2002.

[7] G. Booch, I. Jacobson, and J. Rumbaugh, editors. *Unified Modeling Language, Notation Guide, Version 1.0*. Addison-Wesley, 1998.

[8] Walter Cazzola, Ahmed Ghoneim, and Gunter Saake. Reflective Analysis and Design for Adapting Object Run-time Behavior. In Zohra Bellahsène, Dilip Patel, and Colette Rolland, editors, *Proceedings of the 8th International Conference on Object-Oriented Information Systems (OOIS'02)*, Lecture Notes in Computer Science 2425, pages 242–254, Montpellier, France, on 2nd-5th of September 2002. Springer-Verlag. ISBN: 3-540-44087-9.

[9] A. Charfi and M. Mezini. AO4BPEL: An Aspect-oriented Extension to BPEL. *World Wide Web Journal: Recent Advances on Web Services (special issue)*, 10:309–344, 2007.

[10] 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.

[11] M. Clavel, F. Duran, S. Eker, J. Meseguer, and M. Stehr. Maude : Specification and Programming in Rewriting Logic. Technical report, SRI, Computer Science Laboratory, March 1999. URL : http://maude.csl.sri.com.

[12] C. Courbis and A. Finkelstein. Towards aspect weaving applications. In *Proceedings of the 27th international conference on Software engineering (ICSE '05)*, pages 69–77. ACM Press, 2005.

[13] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services (BPEL4WS 1.1). http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, IBM report, 2004.

[14] C. Dorn and S. Dustdar. Sharing hierarchical context for mobile web services. *Distributed Parallel Databases*, 21:85–111, 2007.

[15] Zi Duan, A. Bernstein, P. Lewis, and S. Lu. A Model for Abstract Process Specification, Verification and Composition. In *Proceedings of the 2nd international conference on Service oriented computing (ICSOC'04)*, pages 232–241. ACM Press, 2004.

[16] A. Erradi and V. Maheshwari, P. Tosic. Policy-Driven Middleware for Self-adaptation of Web Services Compositions. In *ACM/IFIP/USENIX 7th International Middleware Conference*, volume 4290 of *Lecture Notes in Computer Science*, pages 62–80. Springer, 2006.

[17] R. Heckel and L. Mariani. Automatic Conformance Testing of Web Services. In *Proceedings FASE 2005*, volume 3442, pages 34–48. lncs, 2005.

[18] P. Kardasis and P. Loucopoulos. Expressing and Organising Business Rules. *Information and Software Technology*, 2004.

[19] A. Martens. Analyzing Web Service Based Business Processes. In *Proceedings FASE 2005*, volume 3442, pages 19–33. lncs, 2005.

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

[21] J. Meseguer. A Logical Theory of Concurrent Objects and its Realization in the Maude Language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Object-Based Concurrency*, pages 314–390. The MIT Press, 1993.

[22] OMG. UML 2.0: Superstructure Specification. Version 2.0, formal/05-07-04. Technical report, omg.org, 2005.

[23] B. Orriëns, J. Yang, and M.P. Papazoglou. A Framework for Business Rule Driven Web Service Composition. In *Proc. of Conceptual Modeling for Novel Application Domains*, volume 2814 of *Lecture Notes in Computer Science*, pages 52–64. Springer, 2003.

[24] F. Rosenberg and S Dustdar. Towards a Distributed Service-Oriented Business Rules System. In *Proc. of the of EEE European Conference on Web services (ECOWS)*. IEEE Computer Society Press, 2005.

[25] S. Ruy, B. Benatallah, and F. Casati. A Framework for Managing the Evolution of Business Protocols in Web Services. In *In Asia-Pacific Conference on Conceptual Modelling (APCCM'07)*, 2007.

[26] M. Solanki, A. Cau, and H. Zedan. Introducing Compositionality in Web Service Descriptions. In *Proceedings of the International Conference on World Wide Web*. IEEE Computer Society Press, 2004.

[27] W.M.N. Wan-Kadir and P. Loucopoulos. Relating Evolving Business Rules to Software Design. *Journal of Systems Architecture*, 2003.

[28] WFF:. Windows Workflow Foundation. Technical report, http://netfx3.com/content/WFHome.aspx.