Precise and Formal Modelling Methods for Service Systems

Christine Choppy[1] and Gianna Reggio[2]

[1] Université Paris 13 – Sorbonne Paris Cité, France
[2] DIBRIS, Università di Genova, Italy

DIBRIS Technical Report n. TR-14-03

April 2014

1 Introduction

In this report we present three different modelling methods for service systems, but first we introduce our view on what is a service system, by giving a corresponding conceptual model. A modelling method, PreciseSoa, is based on the UML, whereas the other twos are based on the formal logic algebraic notation CASL-MDL, and differ for the descriptive style: property-oriented and constructive, respectively.

The various modelling approaches are presented by applying them to two cases studies.

The structure of the report is as follows. The two case studies are introduced in Sect. 2. Our view on the service system is in Sect. 3, and the three modelling methods are in Sect. 4, 7, and 10 respectively .

2 Case Studies

Here we describe informally two particular service systems, that will be used as case studies for the three modelling methods that we propose. The two cases studies have different features, are not trivial, and are originated by real problems, thus being able to model them is a good validation for our methods.

2.1 Dealer Network

The Dealer Network case study was taken from OMG Adopted Specification of SoaML [1]. The Dealer Network is a business community including three primary parties: the dealers, the manufacturers and the shippers. They are independent parties but they want to work together. Moreover, they have their own business processes and do not want to change their existing systems. A service oriented architecture is required to enable this business environment.

There are three services that will support the working of the parties: Place Order, Get Ship Status and Request Shipping.

The dealers use service Place Order to make an order for the product that they want to buy from the manufacturers. The service supplies the dealers with the quote of the product and issues a confirmation when the order is accepted. If the order is confirmed, the dealer will receive further information about the order, such as the waybill number to track the shipment information.

The manufacturers use service Request Shipping to have the products relative to an order delivered to the dealer by a shipper. The shipper informs the manufacturers of the package picking date and the delivery date. Then, the manufacturers will receive the delivery confirmations when the dealers receive the products.

The dealers use the Get Ship Status service to get information about the status of the shipment for the products they ordered (identified by the waybill number that they received from the manufacturer).

2.2 Office System

Office System has been inspired by the Microsoft Office software. There are various components in the Office System, but we consider only some of them. Our case study includes a set of components supporting office works such as printing, checking spelling and grammar of a given language, and publishing a web page. We name *application* a (not better specified) component using them.

There are the following services to support the work of the office components: Print, Check French, Check English, Check Italian, and Publish on Web.

Service Print provides the applications the means to require the printing of a document; this service can communicate the result of the request, precisely "there is no paper", "the document is not in A4 format", and "the document has been printed". If the paper is not in A4 format, then the user of the service may request either to reduce the page or to cancel the printing.

Three services (Check English, Check French, and Check Italian) allow to check the spelling and the grammar of a text written in a specific language (English, French and Italian). A service of this kind will refuse to do the check, if the text is not in the language it is able to handle. The result of the checking will be the list of the found errors.

Service Publish on Web allows to publish web pages on the Internet. The web page must written in HTML to be published, and the service will refuse to publish a page not written in correct HTML. Then, if the provided URL is correct and the corresponding Web server is available, the service will publish the page, otherwise it will inform of the problems the service user.

3 A View on Services and Service Systems

Fig. 1 presents our view of services and service systems by means of a conceptual model (having the form of a UML class diagram[1]).

[1] In this paper for simplicity we omit the multiplicity equal to 1 in the class diagrams.

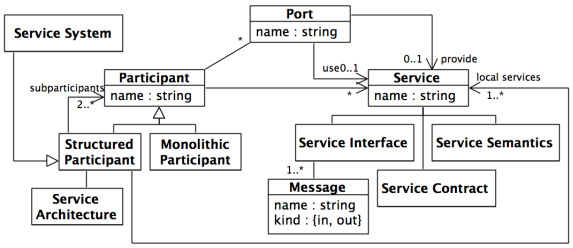

Fig. 1. Service Systems Conceptual Model

The *participants*, i.e., autonomous entities interacting each others by means of services, are the basic constituents of a service system. We classify the participants into structured and monolithic. A *structured participant* includes some inner participants (*subparticipants*), whereas this is not possible for a monolithic participant, but the latter can be still structured, for example in terms of components. Service systems are, then, a special kind of structured participants.

A *participant* is characterized by:

- a number of *ports* (a port represents an interaction point where a participant either uses or provides a service), ,
- a number of services, those that it provides or uses at some of its ports,
- only if it structured, a set of subparticipants, a set of local services, and a service architecture.

A *service system* is then a structured participant neither offering nor requiring services (and thus without any port), so, as said before, a service system is a special case of a (structured) participant.

A *service architecture* presents how the subparticipants of a participant P provide and use their services to allow P to provide its services, obviously taking advantage of the services used by P itself.

A *service* is characterized by a contract, an interface, and a semantics.

The *service interface* provides the static information needed to interact with the service. A service interface is conceptually seen as a set of "in" and "out" messages, were each message is characterized by a name and a list of typed parameters; the "in messages" are used to require the service functionalities to the service provider and the "out messages" to answer to such requests.

The *service contract* focuses on the protocol between the provider and the consumer of the service. The service contract specifies which are the allowed interactions between who uses the service and who provides it, and it may be represented by a labelled transition tree where the transitions are labelled by messages. All the complete paths (i.e., going from the root till a final node) on such tree should start with a transition labelled by an in message. Reaching a final node on such tree means that the service has terminated its activities started as a reaction to the reception of a request from a consumer.

The *service semantics* tries to define which are the functionalities offered by the service. We assume that a service is able to act over a portion of the real world, that we name the *realm* of the service, and that it may modify such realm as the result to having received an in message form a service user, and that its answers to who uses the service (out messages) depend on the current status of the realm. Thus, the semantics of a service consists of a description of the realm, and of a refinement of the labelled transition tree representing the contract, where now the transitions are equipped with conditions on the realm, and with the descriptions of possible modifications of the realm itself.

A *service architecture* defines which are the possible roles for subparticipants within a structured participant (and thus possibly also within a service system), and how the pairs of such (roles for) participants are connected to provide and use services (obviously only if one provides and the other uses the same service), defining also which are these services.

The conceptual schema presented in Fig. 1 could be the starting point to produce an informal model of a participant, that we name *conceptual level model*. It will consists of the lists of ports, subparticipants and services, and of informal presentation of the service constituents (interface, contract and semantics) by means of list of messages and labelled transition trees (where the transitions are labelled by natural language sentences describing receiving or sending messages), of the service architectures (by a graphs whose nodes are labelled by participant roles and whose arcs are labelled by service names), and of the structure and the behaviour of the monolithic participants.

This kind of informal model will be the starting point to prepare a model of the service system following the PreciseSoa method orusing the algebraic specification language CASL4SOA.

Fig. 2 and 3 present by means of two UML activity diagrams the guidelines for helping the modeller to produce a conceptual level model of a participant (and thus, being a special case, of a service system).

4 PreciseSoa: a precise method for modelling service systems using UML

We propose a method to model the service systems using the UML, called PreciseSoa. PreciseSoa adheres to the view of services and service systems presented in Sect. 3, and has been inspired by SoaML (Service oriented architecture Modeling Language), the OMG standard UML profile to architect and model SOA solutions, adopted in May 2012 [1]. Two complete applications of the method can be found respectively in Sect. 5 and 6.

We present the structure of the PreciseSoa models by means of the metamodel shown in Fig. 4, whereas the well-formed constraints are given in Table 1, and the suggested naming policy is reported in Table 2.

PreciseSoa provides two different kind of models for structured and monolithic participants; both offer the possibility to model the services required and provided by a participant; moreover for the structured participants it is possible to

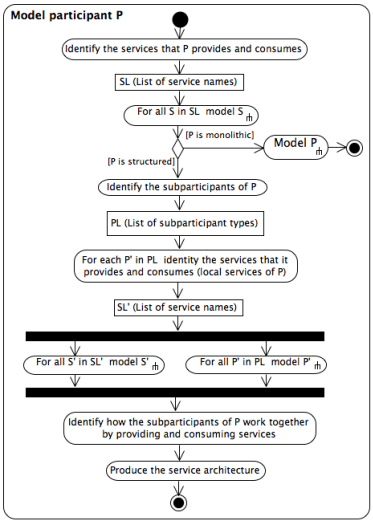

Fig. 2. How to produce a conceptual level model of a participant

define the subparticipants, the local services, and how they are organized into a service architecture, whereas for the monolithic participants the method does not propose a specific way to model their structure and behaviour. Recall that a *service system* is a structured participant neither providing neither consuming services (and thus without ports). As a consequence, we have that a *service system model* is a special case of a participant model.

All used classes and datatypes used in a model must defined within the model itself

4.1 Service Model

A *service model* consists of a name, a service interface, a service contract, and a service semantics. As stated in Sect. 3 a service interface presents the in/out messages that the service exchanges, a service contract specifies an agreement between who provides and who uses the service on how it is provided and used, and the semantics says what are the effects of the received messages on the realm of the service and what will be returned.

A *service interface* is defined by a class stereotyped ≪service interface≫ and named as the service itself. It should realize and use respectively two UML interfaces, defining the in and the out messages by means of operations. The operations of the interfaces correspond to the messages exchanged between the service and the participants using and providing it. The operations may have (in) parameters that must be typed by datatypes (built obviously using only predefined types and other datatypes), and cannot have a return type. The definition of the needed datatypes should be given together with the two interfaces, thus

Service Model
The classes stereotyped by ≪service interface≫ cannot have any operation.

All operations of the service interfaces should have only parameters typed by datatypes, and cannot have a return type.

The collaboration part of a service contract should:
– be named as the service itself,
– have exactly two parts connected by a UML connector, one stereotyped by ≪use≫ and the other by ≪provide≫, and such parts must be typed by the interfaces used and realized by the service interface.

A sequence diagram in the behaviour part of a contract should have exactly two lifelines corresponding to the two parts in the collaboration part of the same contract.

All messages in the sequence diagrams in the behaviour part of a contract of a service should be built by operations of the used and provided interfaces of that service, and all these interface operations should appear at least in one of such sequence diagrams.

Participant Model
All services of a participant must have different names.

All subparticipant types must have different names.

A port of a participant type must be typed by the interface of a service of the same participant.

Service Architecture
All services and participant types of a model should appear in the service architecture part of the same model.

The ports that are connected in a service architecture must be typed by the a pair of conjugate service interfaces.

Table 1. PreciseSoa models: constraints

The names of the roles in a service contract should be written in lower, and should be ending with "er".

The names of the services and of the interfaces should be written in mixed case, where each new word begins with a capital letter, starting with a capital letter.

The names of the operations of an interface should be written in mixed case, where each new word begins with a capital letter, starting with a lower case letter.

The names of datatypes must be substantive, and must written in the mixed case where each new word begins with a capital letter, starting with a capital letter.

The attributes of the datatypes should be written in mixed case starting with a lower case letter.

Table 2. PreciseSoa models: naming policy

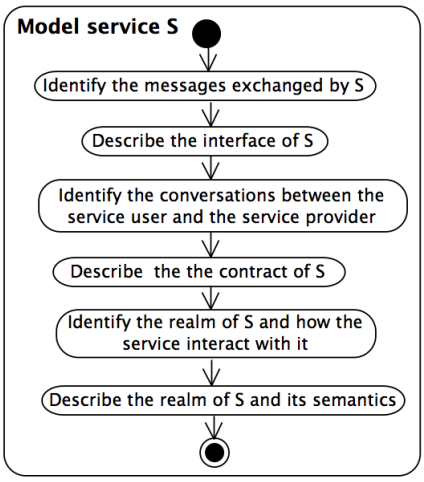

Fig. 3. How to produce a conceptual level model of a service

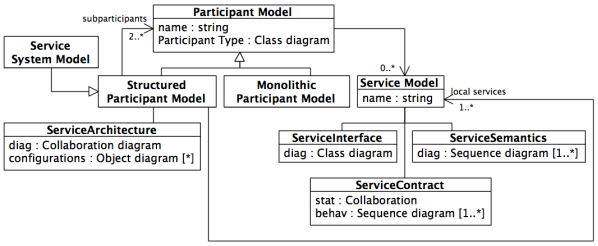

Fig. 4. Service System model: metamodel

(see Fig. 4) a service interface consists of a class diagram, that will include a class stereotyped by ≪service interface≫, the two interfaces and all needed datatypes.

Fig. 5 shows a generic service interface. The class stereotyped by ≪service interface≫ should realize the provided interface (represented by the UML realization symbol: the dashed arrow with closed head) and should use the required interface (represented by the UML dependency: the dashed arrow with open head).

A *conjugate* service interface is suggested as a mechanism to connect the consuming participant and the providing participant. Each service interface has one conjugate service interface that is named by the name of the corresponding service interface starting with " ∼ "; and it is defined transforming the in messages into out messages, and similarly the out messages into in messages, i.e., the realized interface becomes the used one and vice versa.

A *service contract* consists of a UML collaboration stereotyped by ≪service contract≫ and named as the service itself, and by a behaviour represented by

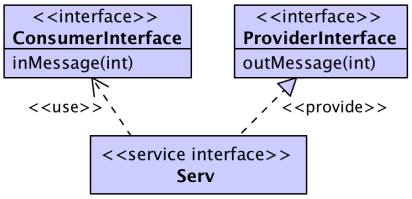

Fig. 5. A generic service interface

a set of UML sequence diagrams. The collaboration has exactly two parts corresponding to the roles the service provider and consumer, and the sequence diagrams have exactly two lifelines (one for the service provider role and one for the service user role).

Fig. 6 shows a generic service contract. The dashed oval is the icon of the collaboration, whereas the inside boxes represent the collaboration parts and are used to model the roles of who provides and of who uses the service (the stereotypes ≪provide≫ and ≪use≫ allow to distinguish the two roles). The parts are typed by interfaces. The sequence diagrams present all possible stories of the provider using the service showing which messages and in which order the provider and the consumer exchange in each story. Thus, in Fig. 6, prov is the role for provider and the ProviderInterface is the interface that it implements to play that role, whereas cons is the role for consumer and ConsumerInterface is the interface that it implements to play that role in Serv. The two parts are connected by a UML connector, to emphasize that they will communicate. Serv is quite simple, and so after receiving an integer number it will return the same number increased by 3, thus a unique sequence diagram is enough to model its contract.

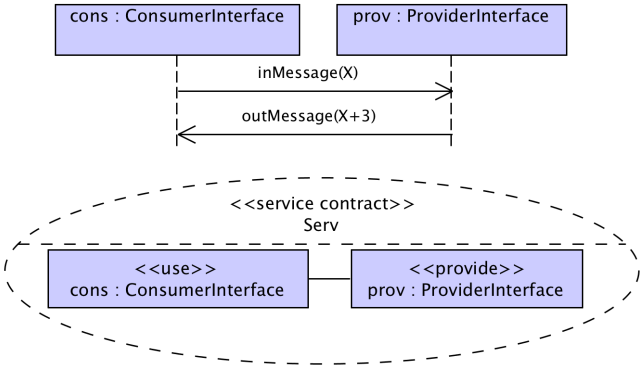

Fig. 6. A generic service Contract

The *service semantics* should be defined by modelling the service realm, how the received messages will result in modifications of the realm status, and how the realm status influences the messages sent out by the service provider. In PreciseSoa we model the service semantics by introducing a class Service_Realm realizing the provided interface, in such a way that its attributes will define the current status of the service realm. Then, the sequence diagrams defining the behaviour part of the service contract, should be refined by adding action specifications to represent the modifications of the realm status and further guards to influence the choice of which messages to send out and which values they are carrying depending on the status of the realm. We do not give a generic example of service semantics.

4.2 Participant Model

In PreciseSoa any kind of participant (monolithic and structured) is described by a specific participant model.

A *participant model* introduces a class stereotyped ≪participant≫, that will be used to type the specific participants (instances) of that kind, and that we will name *participant class*.

A participant is a service provider if it provides a service and is a service consumer if it uses a service. A participant may provide and consume any number of services. It means that the same participant may be a "provider" of some services and a "consumer" of other ones. The UML mechanism of the ports is used to indicate the points of interaction through which participants interact with each others to enact services, and the needed ports are added to the participant class. There are two kinds of port that a participant class may have, one is stereotyped by ≪service≫ where a service is provided, and one is stereotyped by ≪request≫ where a participant makes a request for a service to be provided by other participants. A port is then typed by a service interface. A service port has the type of the provided service interface, and a request port has the type of the conjugate of the interface of the required service.

The inner structure and the behaviour of the monolithic participants will be modelled using the most appropriate UML diagrams, but the PreciseSoa method does not offer any specific indication, mainly because they may have many different forms, for example, they may be structured in terms of components, or they have no structuring at all and their behaviour is defined by an activity diagram.

All the models of the subparticipants of a structured participant (and thus of a service system, that is a particular kind of structured participant) are collect in a *participant view*.

Fig. 7 presents a generic participant view including two participant classes, The instances of the participant class PartX are the provider of Serv and thus the class has a ≪service≫ port, typed by the service interface Serv. Participant class PartY types the consumers of this service and has a ≪request≫ port, typed by the conjugate of the Serv interface (denoted by ∼Serv). These ports are the points for engaging two participants typed byPartX and PartY respectively to enact Serv.

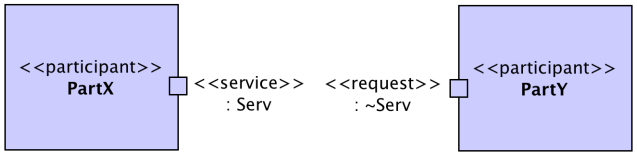

Fig. 7. A generic participant view

4.3 Service Architecture

A *service architecture* is defined by a UML collaboration with stereotype ≪service architecture≫, as shown in Fig. 8, and by a set of *architecure configurations*. A service architecture consists of a set of services and a set of (roles for) participants that work together by providing and consuming services.

The fact that a service is provided and consumed by two participants is represented by a collaboration use of the collaboration part of the contract of such service (e.g., :Serv in Fig. 8). There may be several usages of the same or of different services in a service architecture, and each of them involves a possible different set of roles (and of the related connectors). The collaboration use is decorated by S: Serv (S a name and Serv the service name), where S is optional (as in our example in Fig. 8). The two participants, who play the providing and consuming roles in a service, will be defined in the contract of this service. The roles that the two participants play in a service usage, i.e., who is provider and who is the consumer, are represented by the labels on the dashed line connecting the parts and the collaboration use.

A participant role in the service architecture is displayed as a UML part (a solid rectangle) that contains the optional role name and the participant class typing the role, e.g., : PartX in Fig. 8.

Fig. 8 shows a generic service architecture for a service system, i.e., a structured participant without any port, and thus unable to interact with the outside world by means of service calls.

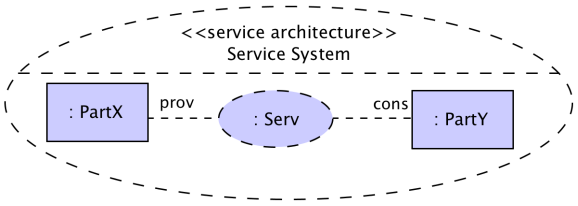

Fig. 8. A generic service architecture for a service system

In case of a structured participant P providing and using services there will be also special parts, denoted by dashed boxes, representing the roles of the participants external to P using or providing services to P.

The service architecture of a generic structured participant is illustrated in Fig. 9. In this case, the roles of the subparticipants are shown as usual by box (e.g., : Part X), whereas the roles of the external participants providing or using the services of StructuredParticipant are shown by dashed boxes (e.g., extProv: PartZ).

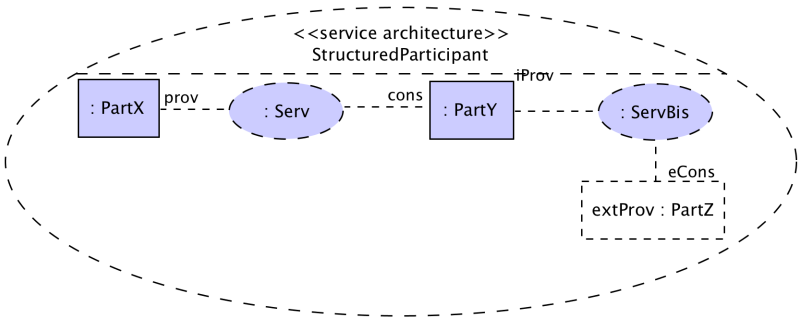

Fig. 9. A generic service architecture for a structured participant

An *architecture configuration* shows a snapshot of a service architecture at a specific point in time. A configuration is presented by a UML object diagram. It includes a set of participant instances and the services that they provide and use at that particular time. Each service is is represented by a use of the collaboration part of the definition of its contract. The links among them are bindings of the collaboration.

4.4 How to produce a PreciseSoa model

Here, we present some guidelines to produce a PreciseSoa model of a participant and thus also of a service system (recall that a service system is a particular case of a structured participant) summarized by the activity diagrams in Fig. 10 and 11. These guidelines refine those presented in Sect. 3, and in Fig. **??** the refined parts are shadowed. In general, we suggest first to produce an informal conceptual model (see Sect. 3), and later starting from it to produce the PreciseSoa model.

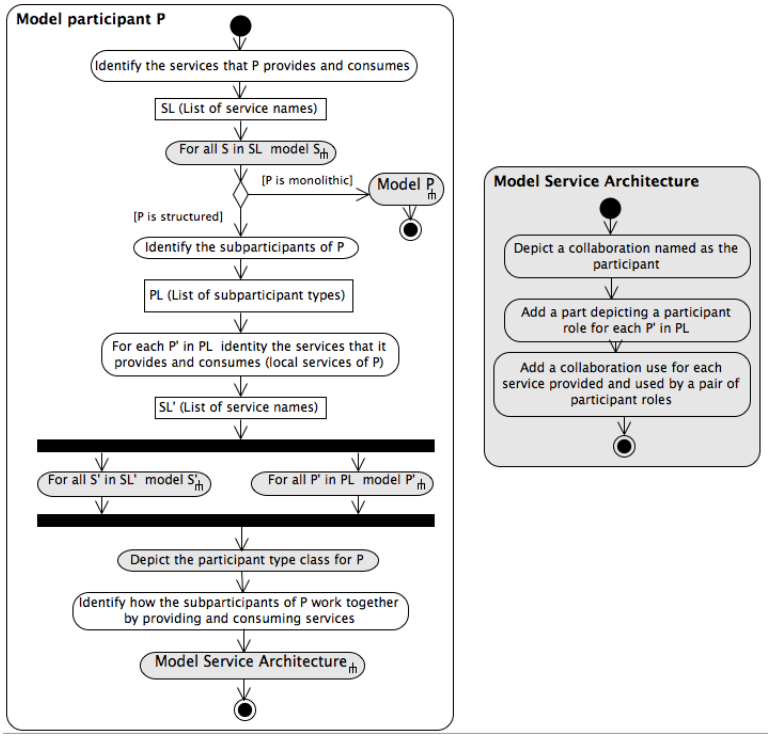

Fig. 10. How to develop a PreciseSoa participant model

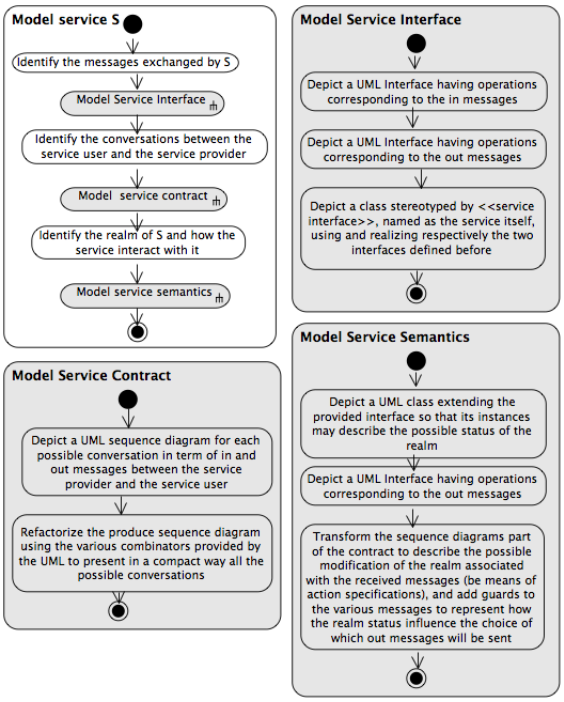

Fig. 11. How to develop a PreciseSoa service model

5 Modelling **Dealer Network System** following **PreciseSoa**

In this section we present the model of the service system Dealer Network System produced following the PreciseSoa method.

5.1 Service Architecture

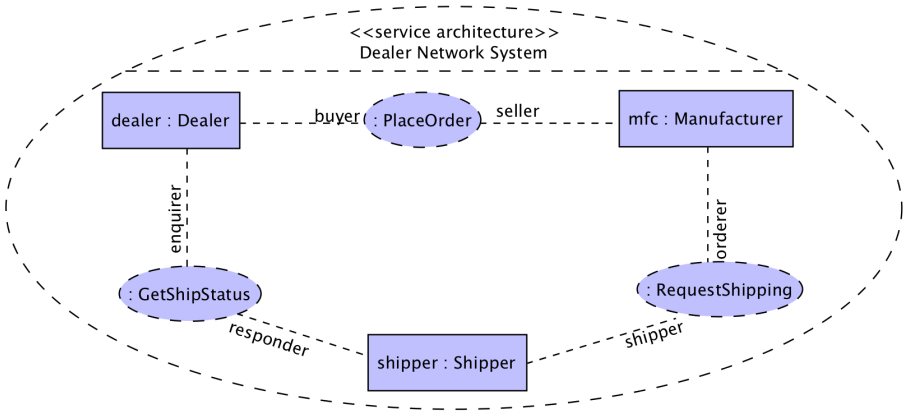

Fig. 12. Dealer Network System: Service Architecture

The service architecture of the Dealer Network System is shown in Fig. 12 and in 13 (which presents an architecture configuration). The Dealer Network System architecture depicts a community of participants providing and consuming services for realizing the aims of the Dealer Network. There are three roles for the participants in this architecture: dealer, shipper and mfc typed respectively by the participant classes Dealer, Shipper and Manufacturer, they are involved in three services: "Place Order", "Get Ship Status" and "Request Shipping". dealer plays the role buyer (i.e., consumer) and mfc plays the role seller (i.e., provider) in service Place Order. Instead, dealer plays role of the enquirer in service Get Ship Status whose provider is the shipper. mfc plays the role of provider in service Place Order, but in service Request Shipping, it plays a role as a consumer - orderer precisely.

Fig. 12 illustrates the possible roles for participants in the high level view of how they work together in the Dealer Network System. The three services and the participants appearing in this diagram will be described by service and participant models in the following subsections. Fig. 13 shows instead a possible configuration of this architecture, where several participants of several types play various roles using and providing services in a particular instant in the life of the system; notice how at the same time several participants may use some service, provided by the same or by other participants.

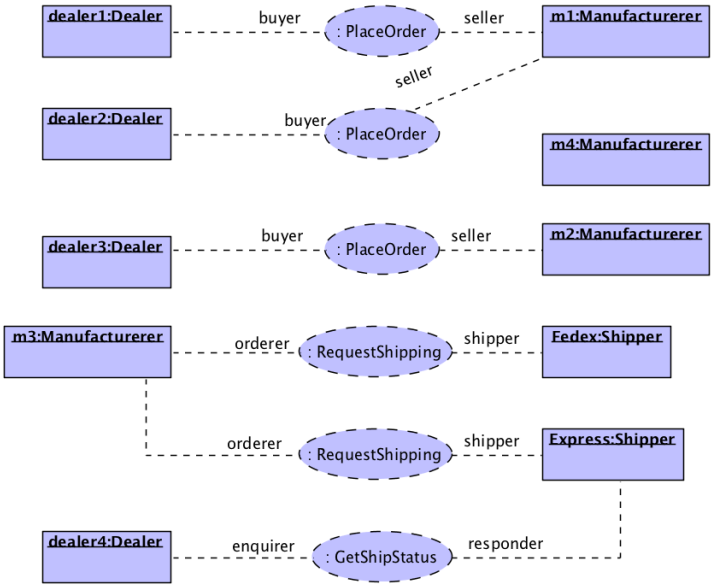

Fig. 13. A Dealer Network System Architecture Configuration

5.2 Local Services

The Dealer Network System has three local services, precisely: "Place Order", "Get Ship Status" and "Request Shipping", whose models are given in the following.

Service Place Order Fig. 14 shows the interface of the service Place Order. It realizes and uses respectively the interfaces: OrderPlacer and OrderTaker, which define the operations that provider and consumer implement to play their own roles. The type of the provider role: OrderTaker is the interface including the operations whose calls the provider seller will receive when enacting the service. The type of the consumer role: OrderPlacer is the interface including the operations whose calls the buyer will receive (correspondingly sent by the provider). The reciver of the order may confirm it, and the fact that an order is confirmed or not is recorded in the boolean attribute confirm of the datatype OrderStatus. Moreover, if the order is accepted, the buyer will receive further information about the order represented by other attributes of the datatype OrderStatus, they are the delivery date and the way bill number of the shipment. The buyers are identified by elements of CustomerID.

Fig. 15 presents the Place Order contract. The collaboration states that the role for the participant using the service is named buyer and is typed by the interface OrderPlacer, whereas the role for the participant providing the service is named seller and is typed by interface OrderTaker. Those parts are bound to fulfill service Place Order following its contract. The sequence diagrams describe how

to use the service, i.e., which messages to send and which may be the possible answers. The buyer may either send a quote request or place an order, and the two sequence diagrams model these two cases.

Fig. 16 finally shows the Place Order semantics. The realm of the service is characterized by the amount of product in stock, so the PlaceOrder_Realm class has an attribute modelling the stock amount stock: int. The modified sequence diagram for the case of placing an order shows that the order is confirmed only whenever the ordered quantity is in stock, otherwise is cancelled, and after an order has been accepted the stock is reduced by the ordered amount.

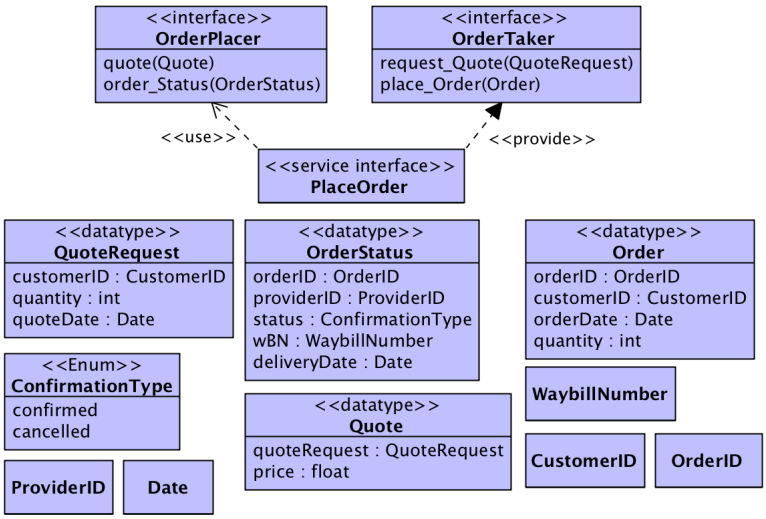

Fig. 14. Place Order model: interface

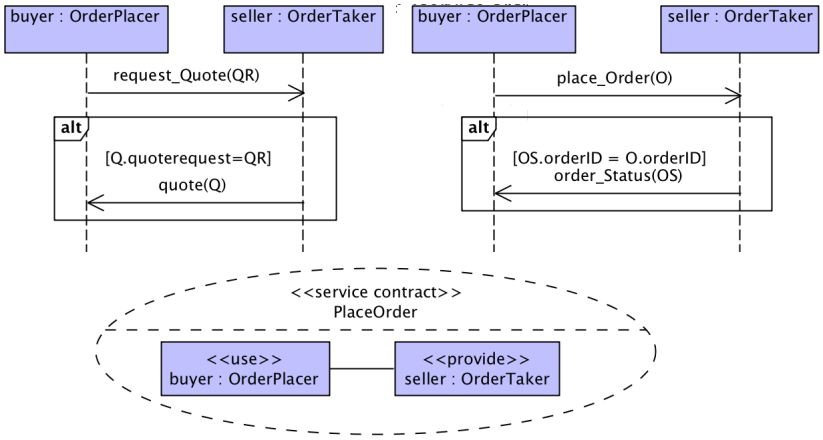

Fig. 15. Place Order model: contract

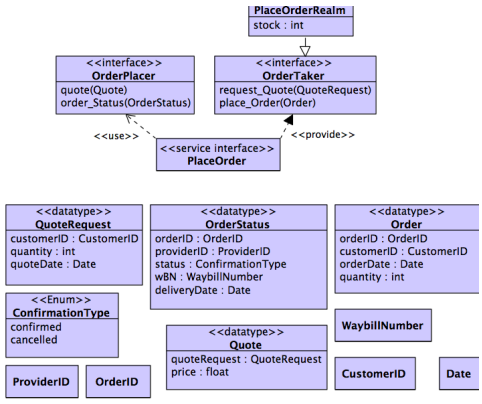

Fig. 16. Place Order model: semantics

Service Request Shipping Service Request Shipping provides the capability to send a shipping request to a shipper in order to deliver goods to a customer for a filled order.

Fig. 17 shows the interface of the Request Shipping service. The provided interfaced contains a unique operation (request_Shipping(Request)), i.e., the service has a unique in message, that will be used to request a shipping (contain the information of the order, i.e., this is waybill number, and the sender and receiver's address); whereas the required interface contain two operations, i.e., package_PickUp(PickUpInfo) and confirm_Delivery(Confirmation), corresponding to two out messages to communicate the info on the pick up (pickup date and an estimated delivery date) and to confirm the delivery (containing the delivery date). Some obvious conditions relate the estimated delivery date, the expected delivery date, the pick up date and the request date can be seen in several dataypes types near the Request Shipping interface.

Fig. 18 shows the Request Shipping contract. In the sequence diagram to ensure that the package pickup message is sent to corresponding shipping request, we require that the parameters R and PUInfo in satisfy the condition R.wBN = PUInfo.wBN, similarly for R and conf.

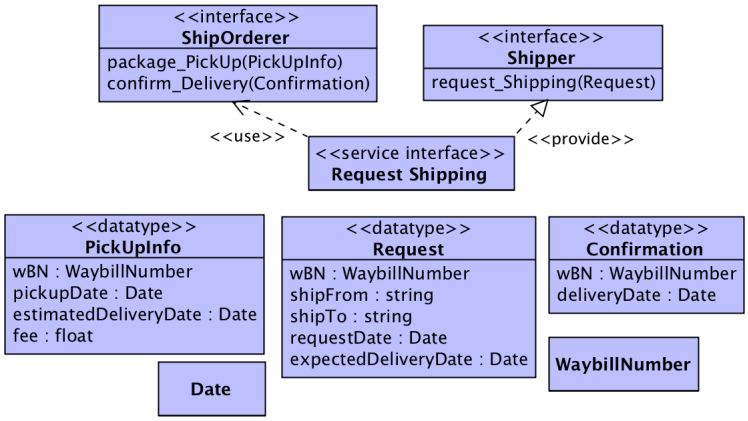

Fig. 17. Request Shipping model: interface

We do not give the semantics of the service Request Shipping since to know exactly why the shipper proposes a specific date for the pick up is of no interest for the service user, while it is clear that who requires the shipping cannot cancel the request once s(he) knows the delivery date.

Service Get Ship Status The interface and the contract of the service Get Ship Status are shown in Fig. 19 and 20 respectively; similarly to the the service Request Shipping we do not give its semantics.

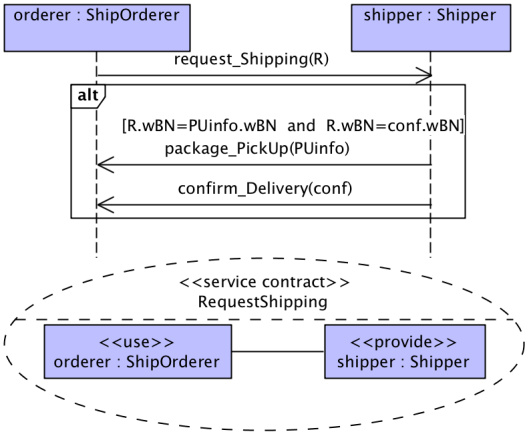

Fig. 18. Request Shipping model: contract

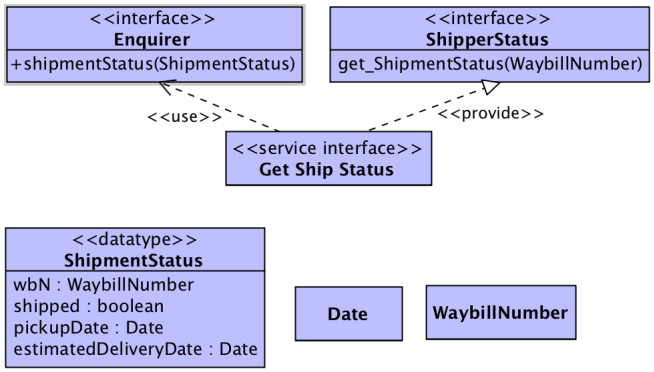

Fig. 19. Get Ship Status model: interface

The collaboration in Fig. 20 binds the two parts representing the consumer and provider of service Get Ship Status. The consumer plays a role as a enquirer that is typed by interface Enquirer (defined in Get Ship Status interface in Fig. 19). Enquirer includes an operation to receive the status of the shipment, i.e., shipment_Status(ShipmentStatus).

The type of the provider role responder is ShipperStatus interface that contains one operation get_ShipmentStatus(WaybillNumber) called by the consumer to enact this service. If we could not find any suitable names for the roles, we may let them have the same names as the interface themselves (shipperStatus: ShipperStatus for example). The purpose of the service is resulting shipment status for the enquirer. The status of the shipment will be contained in the datatype ShipmentStatus (see Fig. 19). The order of messages that service Get Ship Status receives and sends out is illustrated by the sequence diagram appearing in Fig. 20.

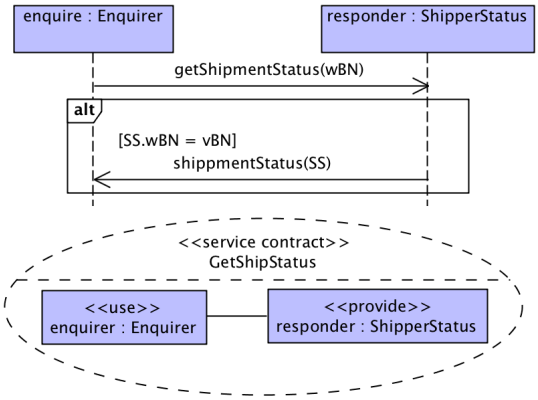

Fig. 20. Get Ship Status model: contract

5.3 Participant View

All the kinds of participants that provide and consume services in the Dealer Network System are presented by the participant view show in Fig. 21 giving only their participant classes (here we do not further model these participants, not even we express if they are structured or monolithic). There are three kinds of participant: Dealer, Manufacturer, and Shipper. Each participant (type) has service ports and request ports for the services they provide and consume that are stereotyped by ≪service≫ and ≪request≫. Manufacturer participant is a service provider for service Place Order, so it has a ≪service≫ port typed by the service interface PlaceOrder to provide the service through this port. Manufacturer is instead a consumer of service Request Shipping, then it has a ≪request≫ port typed by the conjugate of the Request Shipping interface (∼Request Shipping).

Similarly, Dealer is a consumer of the service Place Order provided by Manufacturer.

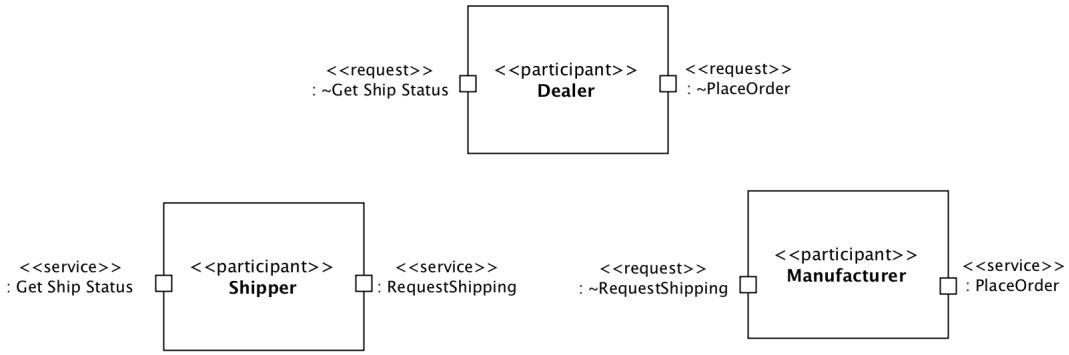

Fig. 21. Dealer Network System: participant view

6 Modelling **Office System** following **PreciseSoa**

In this section we model the Office System described in Sect. 2 following the PreciseSoa method introduced in Sect. 4.

6.1 Service Architecture

The service architecture of Office System shown in Fig. 22 presents the participant roles of the system and which service they provide and consume. There are six roles for participants: typed respectively by OfficeComponent, PrintingCenter, EnglishCenter, ItalianCenter, FrenchCenter, and WebPublisher. In this architecture, the participants typed by OfficeComponent play the consumer role to the services: Print, Check French, Check English, Check Italian, and Publish on Web. Fig. 23 shows instead a possible configuration of the Office System.

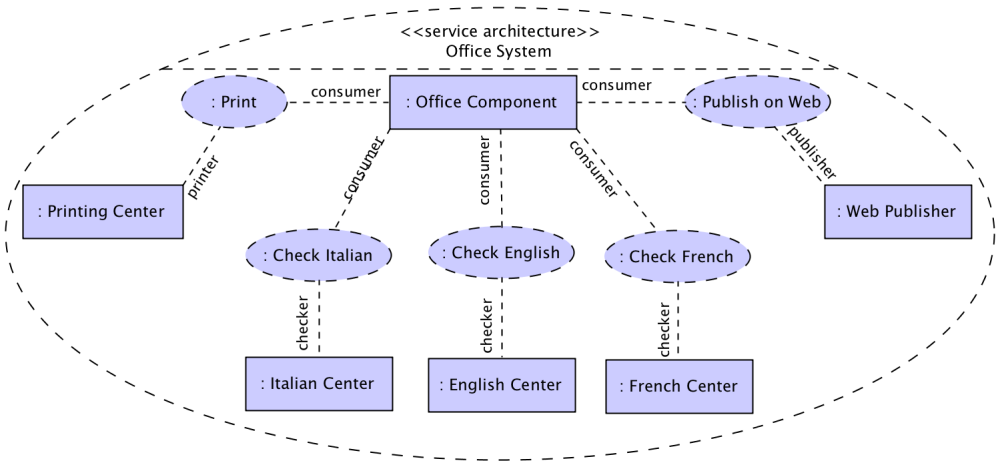

Fig. 22. Office System: service architecture

6.2 Local Services

Service Print The purpose of the Print service is to allow to print documents.

Fig. 24 shows the interface of Print. Fig. 25 depicts the Print service contract, where the two roles writer and printer, typed respectively by the interfaces Writer and Printer, are the consumer and the provider respectively. The Printer interface contains three operations to enact the service, as shown in Fig. 24. The interface Writer instead contains three operations corresponding to the three messages: printed if the document printed successfully, notA4 if the page is not in A4 forma,t and noPaper if the printer is out of paper. The messages exchanging between the

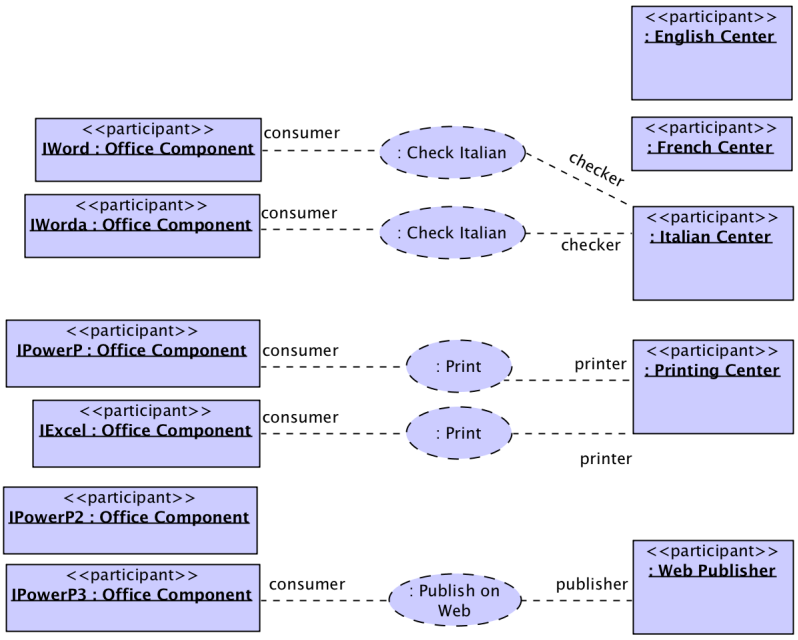

Fig. 23. A Office System architecture configuration

parts that provide and use that service is represented by the sequence diagram in Fig. 25. Fig. 26 shows the semantics of the Print service; here we can see, e.g., that the message noPaper is truly motivated by the lacking of paper, and that it is sent also in the case there is paper sufficient to print a fraction of the document.

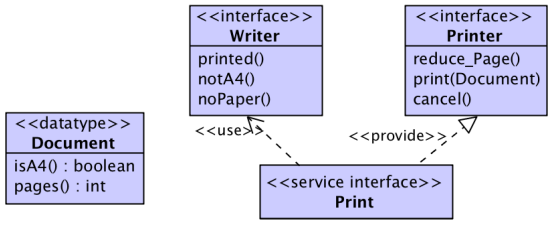

Fig. 24. Print model: interface

Check Italian (Check French, Check English) service There are three services that supply capabilities to check the spelling and the grammar of a text written in one of three specific languages: English, French and Italian (as

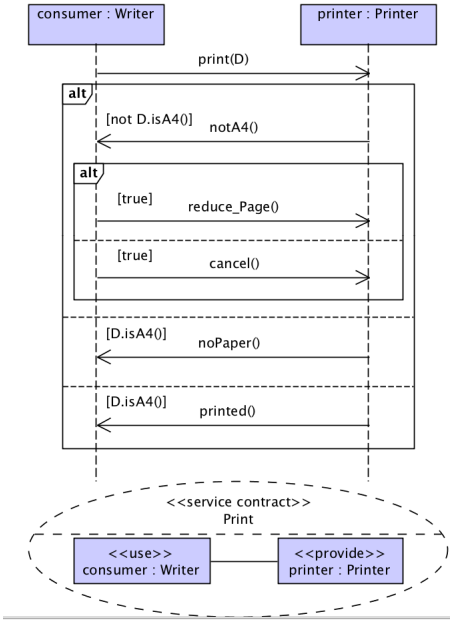

Fig. 25. Print model: contract

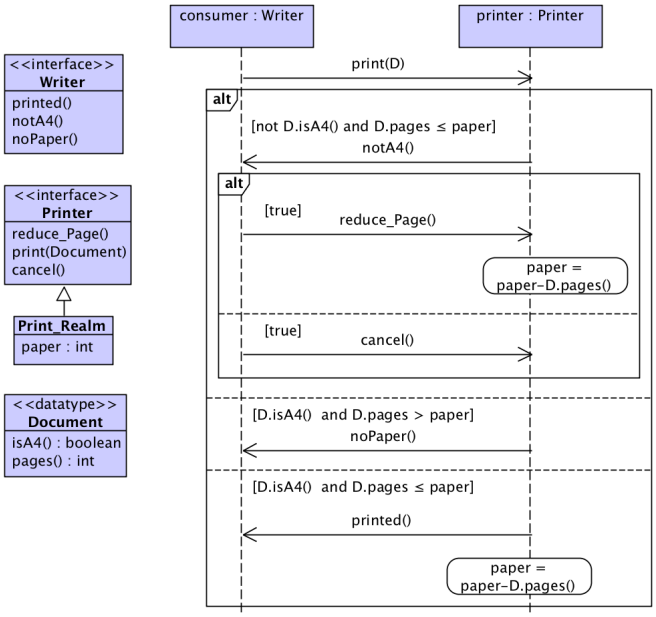

Fig. 26. Print model: semantics

described in Sect. 2). Here we give the model of the service Check Italian considering the Italian language, the models of Check French and Check English are similar.

The interface of Check Italian is given in Fig. 27. The provider interface Checker comprises two operations, i.e., check_Spelling(Text) and check_Grammar(Text), they are two types of in message that this service can receive, one for spelling checking, and the other for grammar checking. The user interface Editor comprises three possible out messages that service may send out, i.e., spelling_Errors(SpellErrors) if the service found any spelling error in the submitted text, grammar_Errors(GrammErrors) if the service found any grammatical error in the text, and wrong_Language() if the text is not written in the Italian language.

The contract of the service Check Italian, see Fig. 28, states that if it receives a message check_Spelling(T), then it may return only one of the two messages: spelling_Errors(...), wrong_Language(), while if it receives check_Grammar(T), then it may answer also grammar_Errors(...). We give two sequence diagrams for this service contract to illustrates those two possible cases, see Fig. 28.

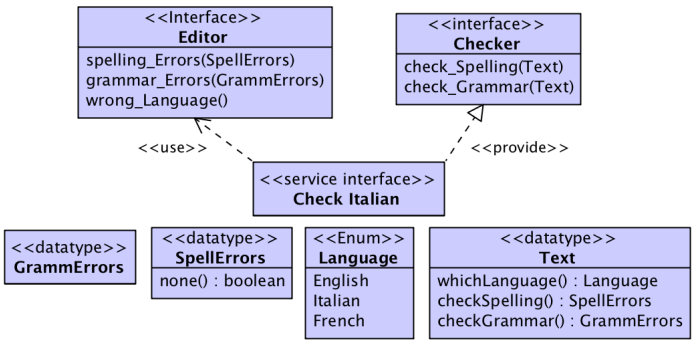

Fig. 27. Check Italian service: interface

We do not give the semantics of the service Check Italian, since this service does not depends on a realm and does not modify anything; moreover, there is no sensible way to specify the correctness of a text expressed in the italian language, and also if possible it will be of no use to anyone.

Service Publish on Web The consumers use the Publish on Web service to publish web pages on the Internet.

Fig. 29 shows the interface of service Publish on Web. The role for the participant using the service is named consumer and the role for the participant providing the service is named publisher. In the collaboration in Fig. 30, they appear again in two parts of the service contract and are bound to their types, i.e. the Web Editor and Publisher interfaces respectively, as shown in Fig. 29. The operations that are sent from the publisher to the consumer can be: wrongURL()

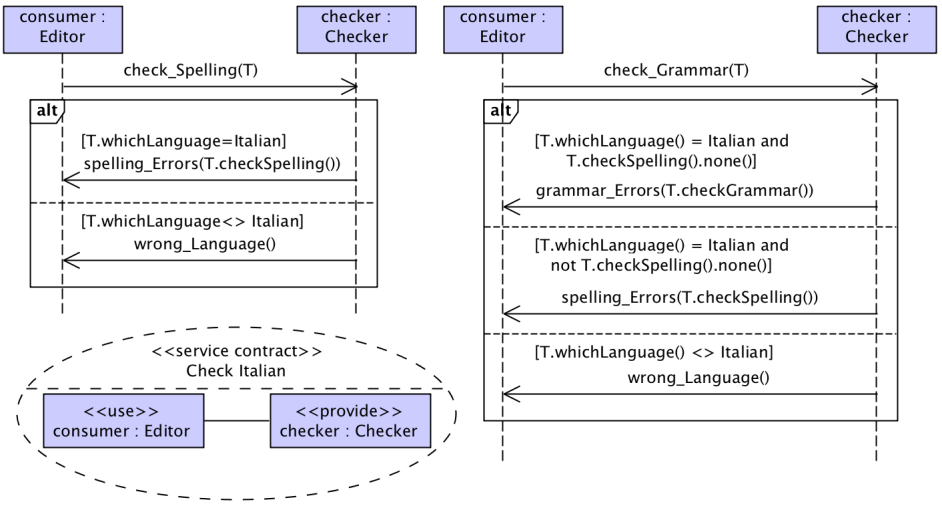

Fig. 28. Check Italian service: contract

if the URL is wrong, notHTML() if the page is not written in HTML, serverNotAvailable() if the requested server is not available at this moment, and published() if the page has been published. Fig. 30 illustrates this service contract.

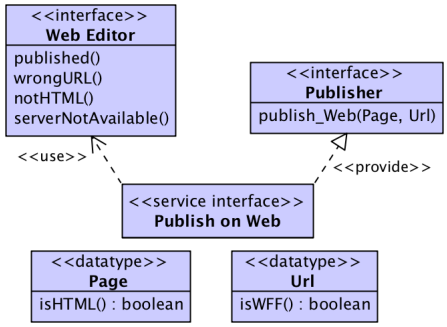

Fig. 29. Publish on Web service: interface

6.3 Participant View

There are six kinds of participants that provide and consume services in Office System. They are represented by the classes stereotyped by ≪participant≫ in the participant view in Fig. 31 and named: ItalianCenter, EnglishCenter, FrenchCenter, PrintingCenter, OfficeComponent, and WebPublisher. Those participants realize their interfaces by service ports and request ports that are stereotyped by

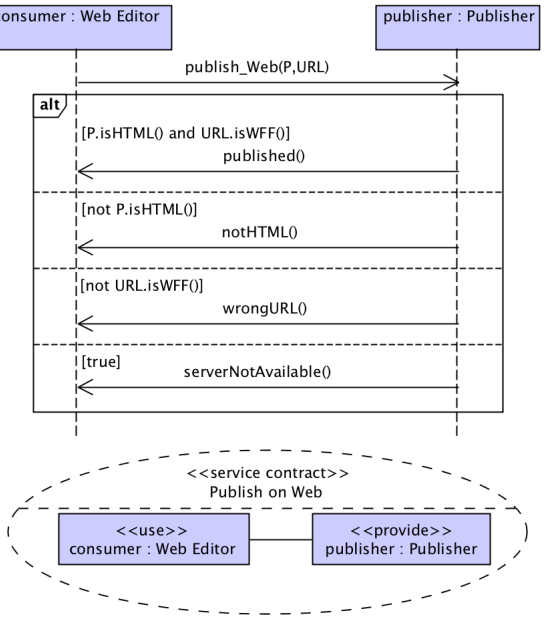

Fig. 30. Publish on Web service: contract

≪service≫ and ≪request≫. The participant typed OfficeComponent is the unique consumer and does not play any provider role in the system, since all its ports are stereotyped by ≪request≫ and typed by the conjugate service interfaces of the providers for corresponding services. For the participants that are of kind of providers, their ports are stereotyped by ≪service≫ and typed by the service interface of the service itself. The participant typed PrintingCenter is provider for the Print service, its port is stereotyped by ≪service≫ and typed by the service interface Print.

7 Design Model of Service system in Casl4Soa

In this chapter, we present our extension of CASL-MDL models that offers a visual syntax to a subset of the CASL-LTL [**?**] formal textual specifications to develop CASL4SOA as a formal visual notation used to model service systems not only visually but also formally and effectively.

8 Overview of Casl4Soa

Based on CASL-LTL (an extension for dynamic systems of the algebraic specification language CASL, see [**?**]), CASL-MDL [**?**] has been developed as a non objected-oriented visual formal notation. CASL-MDL offers a visual syntax to a

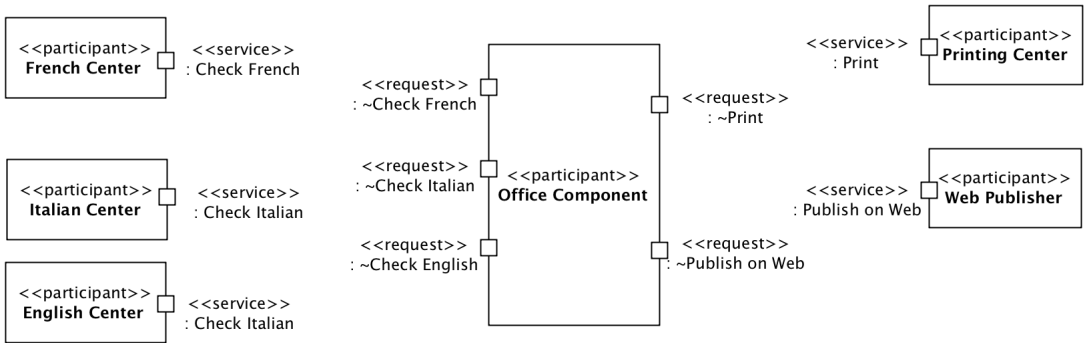

Fig. 31. Office System model: Participant View

subset of the CASL-LTL formal textual specifications, precisely each CASL-MDL model can be translated into a CASL-LTL specification.

In CASL-MDL we have a type diagram introducing the datatypes and the dynamic types, which are types of dynamic systems, that allow to model datatypes and either simple or structured dynamic systems.

A dynamic system is seen as a labelled transition system, where the labels are "elementary interactions" corresponding to the interactions of the system with its context. Then, the behaviour of data and dynamic types is defined either following a property-oriented style using logical formulas (of a first-order many sorted branching time with edge-formulas logic) or constructively defining the operation behaviour by conditional rules and the system behaviour by interaction machines.

At the visual level the constructs of CASL-MDL have been defined reusing the visual constructs of the UML, thus we can use the UML editing tools to produce the CASL-MDL models.

CASL4SOA [**?**] (Common Algebraic Specification Language for Service Oriented Architecture) has been developed as a profile of CASL-MDL with the aim to provide an effective notation to model SOA systems. The profiling mechanism used for defining CASL4SOA is similar to the profiling mechanism of the UML, and it was inspired by it. Thus we use stereotyped CASL-MDL constructs (Appendix **??**) to define the new CASL4SOA constructs. Moreover, each CASL4SOA model corresponds to a CASL-MDL model, that in turn corresponds to a CASL-LTL specification (see Fig. 32), which has a well-defined formal semantics, thus also CASL4SOA has a well-defined formal semantics.

CASL4SOA has been designed over SOA paradigm presented in Chapter 3, and here we will use all the terminology defined in such chapter.

There are two kinds of CASL4SOA service models, constructive and property-oriented. In a constructive model, the behavioural aspects of the services and of the participants are expressed by means of the interaction machines (see Appendix **??**), whereas in a property oriented model, such aspects are expressed by means of first-order temporal logic formula (see Appendix **??**).

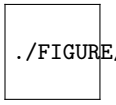

Fig. 32. Relationships among Casl4Soa, Casl-Mdl, and Casl-Ltl

In Casl4Soa, *dynamic system* denotes any kind of dynamic entities, i.e., entities with dynamic behaviours without making further distinctions, and are formally considered as labelled transition systems, that we briefly summarize below.

A *labelled transition system* is a triple $(State, Label, \rightarrow)$, where $\rightarrow \subseteq State \times Label \times State$ is the *transition relation*.

A dynamic system is thus modelled by a transition tree determined by a labelled transition system and an initial state $s_0 \in State$. A dynamic type corresponds to the state of a labelled transition system, thus its values correspond to dynamic systems.

The labels of the transitions of a dynamic system are named *interactions* and are descriptions of the information flowing in or out the system during the transitions, thus they truly correspond to interactions of the system with the external world. The states of simple systems are characterized by a set of typed attributes (precisely the states of the associated labelled transition system).

We use dynamic types to model services and participants.

9 Casl4Soa Constructive Service System Model

The form of the Casl4Soa constructive participant models is shown in Fig. 33 by means of a metamodel, whereas Table **??** shows the constraints defining the well-formed models.

A service system, as said in Chapter 3, is a particular structured participant neither offering nor using services, and thus without any service point. Thus a Casl4Soa *constructive service system model* is a special case of a participant model for a participant without any port, thus neither offering nor using services.

A participant model consists of the definition of a participant type by means of a type diagram including the definition of a dynamic type, and of the models of the provided and used services. The dynamic type defining the participant may be either simple (for the case of the monolithic participants) or structured (for the structured participants), in this case the structured dynamic type definition will allow to represent its service architecture. The model of a structured participant will include also the models of a set of *participant types*, its subparticipants, and the models of the local services, i.e., the service used by its subparticipants to interact among them.

A service system, as said in Chapter 3, is a particular participant neither offering nor using services, and thus without any service point. Thus a Casl4Soa

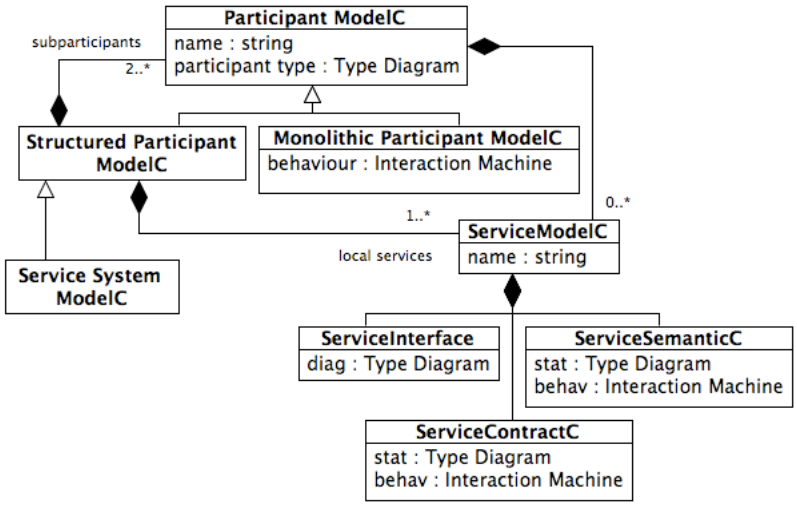

Fig. 33. CASL4SOA constructive model: metamodel

constructive service system model is a special case of a participant model for a participant without any port, thus neither offering nor using services.

The CASL-MDL dynamic type modelling a participant will be stereotyped by ≪Participant≫, whereas in the case of service system the stereotype ≪ServiceSystem≫ will be used.

Participant Model

– All services have different names.

– All subparticipants have different names.

– A subparticipant must have at least a port to offer or use at least one service.

– Connectors between two ports of two different participants must be labelled by the name of a service that those participants offer and use.

– Many different connectors may leave or enter the same port of a participant but all of them must be labelled with the same service.

– The services used to label the connectors must be already presented in the model.

Service Model

– All the parameters of the interactions of a service interface must be typed by datatypes.

– A service interface must have at least one input elementary interaction.

– If a service is named SN, then the simple system appearing in the contract should be named SN_Contract and the one appearing in the semantic should be named by SN_Semantics.

– There is only one initial state in the interaction machine representing service behaviour.

CASL4SOA constructive model: Well-formedness constraints

Naming Convention

–In the names of the services and interfaces, each word should begin with a capital letter (e.g., Place Order, OrderTaker).

–In the names of the interactions, word should be written in mixed case starting with lower case. When there are more than two words in the name, use underscores to separate them (e.g., ?_place_Order).

–The parameters of an interaction should be in upper case (e.g., QR).

–Names of datatypes must be nouns, and each word of the names should begin with a capital letter (e.g., OrderStatus).

–The attributes of datatypes should be written in mixed case starting with lower case (e.g., orderDate).

Table 3. Naming convention for CASL4SOA models

9.1 Service Model

The structure of the service model is shown as a part of CASL4SOA Constructive Model in Fig. 33 (together with Participant Model which is described in Sec.9.2), and the associated well-formedness constraints are reported in Table **??**. A CASL4SOA *constructive service model* consists of the *service name*, a *service interface*, a *contract* and a *semantics*. As stated in Chapter 3, a service interface provides the static information needed to interact with the service, the service contract focuses on the protocol between the provider and the consumer of the service, and the semantics allows to understand the functionalities provided by the service to its users.

A *service interface* is an interface for a dynamic system, visually represented by a box with the stereotype indication ≪Service Interface≫ (see a generic service interface shown in Fig. 34), it is named as the service itself and it defines the elementary interactions needed to use the service, distinguished in input and output interactions by a naming convention ?_yyyy (input) and !_xxxx (output). The input elementary interactions model the requests sent to the service, whereas the output ones model the answers that the service sent out to the user. They are characterized by a name and a possible empty list of parameters.

A *service contract* in constructive style is represented by a simple dynamic system stereotyped by ≪simpleSystem≫ (see a generic simple system shown in Fig. 35) and an associated *interaction machine* (see Fig.36), in which the behaviour of the service is modelled as seen at the service point where it is provided.

The simple system should extend the one used for modelling the interface but without adding any new interaction, thus it will exactly the same elementary

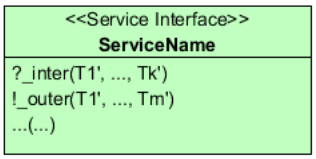

Fig. 34. A generic service interface

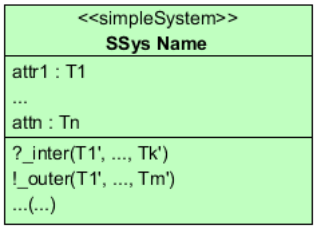

Fig. 35. A generic simple dynamic system

interactions as the service interface, whereas it may have some new attributes, which are needed to abstractly model the relationships between the in and out messages. The interaction machine modelling a service contract should follow a specific pattern to mimic the informal conceptual description of a service contract proposed in Chapter 3 to illustrate the fact that: the service may receive initially possible requests, then it will answer in many different ways (even going in a final state), after that the service is ready to receive other requests, then answer them. A visual generic schematic example of an interaction machine is given in Fig.36, and for further description, see Appendix **??**).

The realm of a service is the part of the real world affected and known by the service itself. The *semantics* of a service describes the effects of the requests received by the service itself (in messages) on its realm, and how the realm status determines the answers sent out by the service itself (out messages). The semantics of a service in constructive style is given by a simple dynamic system extending the one used for the service contract (again without adding new interactions), and the behaviour of this system is modelled as usual by means of an interaction machine.

9.2 Participant model

A participant type is expressed as a CASL-MDL dynamic type (see Appendix **??**) stereotyped by ≪Participant≫ (see generic participant shown in Fig. 37), the ports of which are characterized by service interfaces to indicate that a participant of that type provides or consumes a service. The ports are used to structure the elementary interactions of a dynamic system and to define the cooperation inside the structured dynamic systems.

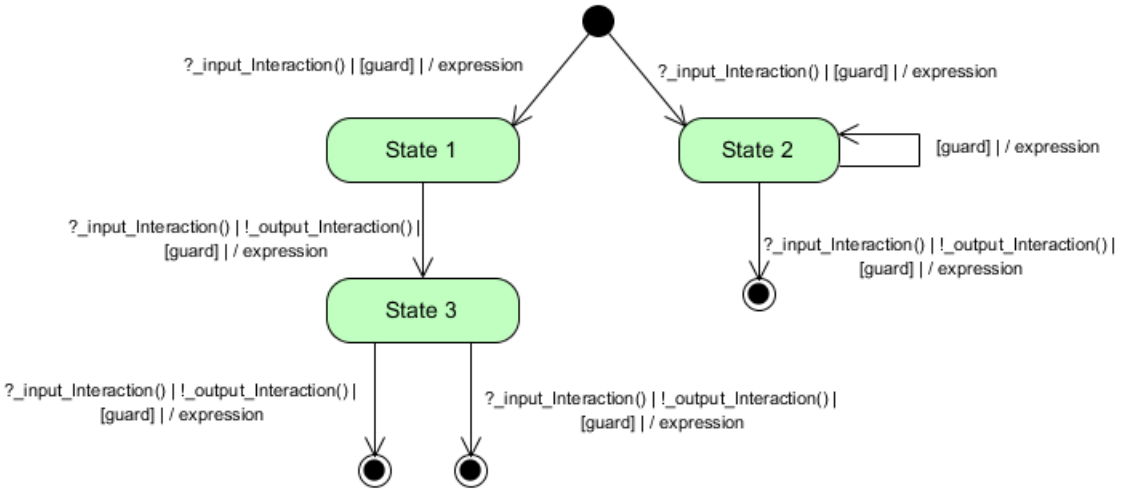

Fig. 36. A generic schematic example of an interaction machine

If a participant is offering a service S, the port where S is offered is typed by the interface of S, whereas if it is using a service S' throughout a port, then such port is typed by the conjugate of the interface of S'. Recall that the conjugate of an interface I is denoted by ∼I, and the elementary interactions of ∼I are those of I where the input and output types are swapped.

The participant type and all the needed datatypes are collected in a type diagram.

In case of a monolithic participant, a generic model is given in Fig.37, its behavior should be expressed by means of an interaction machine (a generic interaction machine is given in Fig.36).

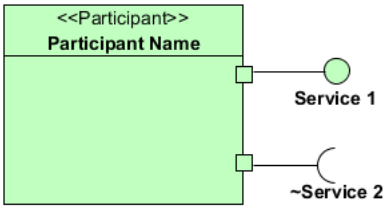

Fig. 37. A generic participant

In case of a structured participant, the corresponding CASL-MDL dynamic type will be a structured dynamic type (see Fig.38), and its definition will allow to express the service architecture of the participant itself.

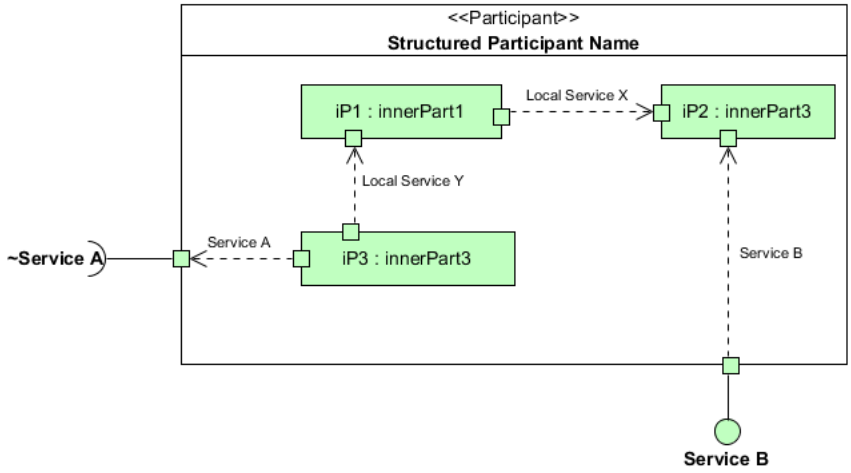

Fig. 38. A generic structured participant

As described in Chapter 3 a service system is a special case of structured participant without any port, and we will use the stereotype ≪ServiceSystem≫ (see Fig.39) instead of ≪Participant≫ for denoting the corresponding CASL-MDL structured dynamic type.

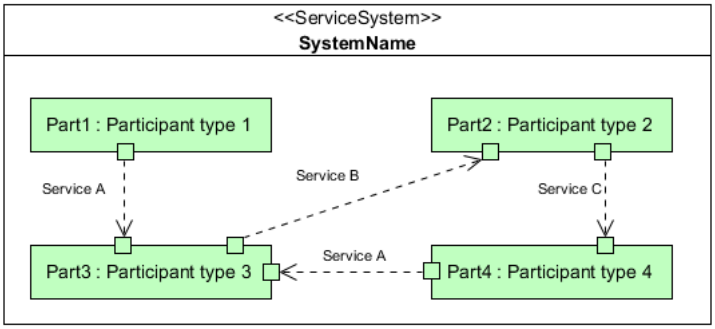

Fig. 39. A generic service system

10 Casl4Soa property oriented model

The structure of the Casl4Soa property oriented models is shown in Fig. 40. The form of these models is similar to the one of the constructive models shown in Fig. 33, but now the behaviour of the dynamic systems is modelled by means of a set of logical formulas, instead of an interaction machine. The constraints defining the well-formed models are the same as the constructive models, shown in Table **??**.

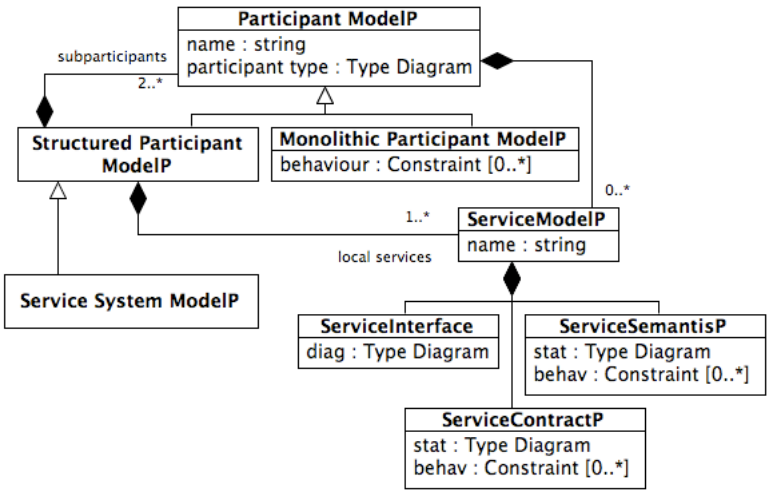

Fig. 40. Casl4Soa property oriented model: metamodel

A *service contract* in property-oriented style is represented by a dynamic type whose behaviour is specified by a set of constraints on the type itself, i.e., by a set of temporal formulas described in Appendix **??**. Similarly the semantics of a service in property-oriented style is given by a simple dynamic system extending the one used for the service contract, and the behaviour of this system is modelled again by means of a set of constraints.

A generic example of a formula is presented as following:

[in_any_case] [sometimes — always] [path_form] ⇒ *[eventually — always — next][path_form]*

The formulas comprise first-order logic combinators, together with temporal combinators (for a path formula) to address whether a property is satisfied in states of a path from a given state. The form of formulas should conform to a grammar structure defined in Appendix **??**.

11 How to develop a Casl4Soa model

Following the indications of Chapter 3 (see Fig. **??** and **??**), we give first a conceptual model of the service system of interest, and then model all the parts using CASL4SOA; all the steps are summarized in Fig. 41.

Recall that a service system is a particular structured participant neither offering nor using services, we assume that the service system being built is the first considered structured participant of type P (see activity named **Model participant type P** in Fig. 41), then it may include a number of monolithic participants and structured participants, and a number of services.

Identifying the services that the subparticipants of P provide and consume is the first steps.

To model a service S in CASL4SOA, all the steps are shown in the actions of the activity named **Model service S** in Fig. 41. Those actions are described in details as following:

Give the simple dynamic type representing the interface of S

What to do

Define set of input and output interactions for a simple dynamic system representing interface of S. The interactions are either of kind input or output. Define datatypes needed to type the elements of interface.

What is supposed to be clear before building interface of S

- The specific requests that service S will receive from the users.
- The specific responses that service S may send to the users for corresponding requests.

The composition of S interface

- Name of S interface
- The elementary interactions of kind *input* and *output*.

Building steps

1. Use CASL-MDL simple dynamic type with stereotype ≪serviceInterface≫ to define S interface.
2. Name S interface by the name of S itself (suggesting the purpose of the service)
3. Build input interactions based on the requests to service S, give them names (starting with convention ?_) and define the necessary parameters.
4. Build output interactions based on the requests to service S, give them names (starting with convention !_) and define the necessary parameters.
5. Use construct Datatype to define the elements of S interface if they are not of primitive type.

Note:

- The name of a input interaction should contain and start with a verb.
- The input interactions should be listed before the corresponding output interactions.

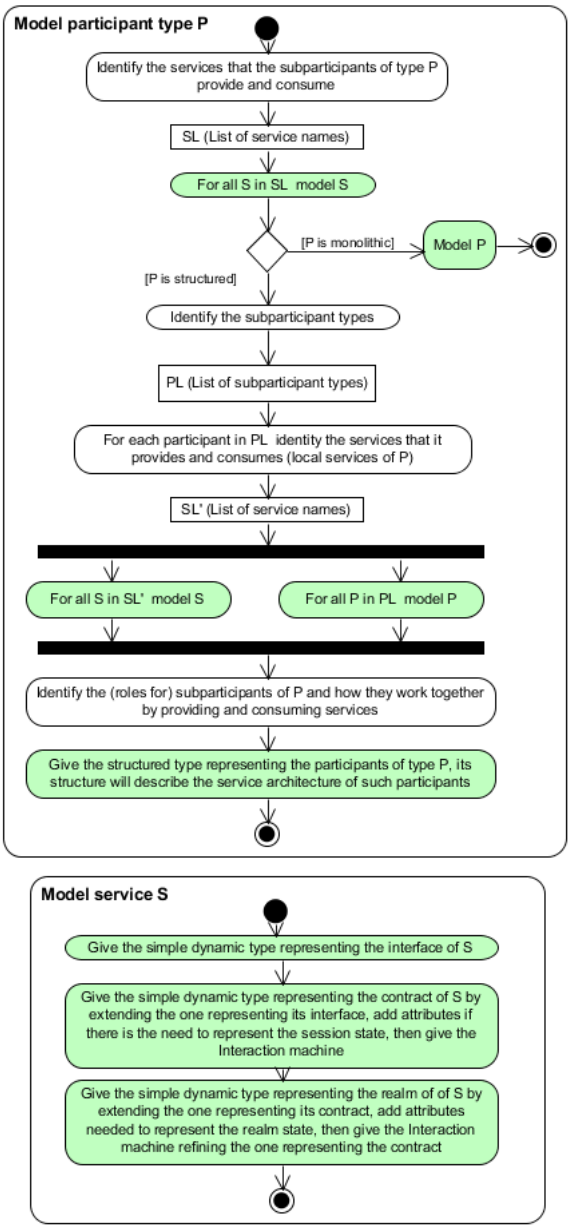

Fig. 41. How to develop a CASL4SOA model

Give the simple dynamic type representing the contract of S
What to do

Extend the simple dynamic type that represents the interface of S, add attributes if needed to represent the session state.

What is supposed to be clear before building the contract of S

- The interface of S.
- The attributes needed to model the activities of S providers and to express the realm of the service.

The composition of S contract in terms of a simple dynamic type

- Name of S contract (named with extension _Contract).
- Some attributes determining the internal states of the dynamic system representing S contract, and the elementary interactions as defined in S interface.

Building steps

1. Use CASL-MDL simple dynamic type with stereotype ≪simpleSystem≫ to define S contract.
2. Name the simple dynamic system by S name with extension _Contract, and insert the elementary interactions that are predefined in the interface of S.
3. Insert attributes representing the session state and type them by datatypes/primitive types.

Give the interaction machine of S contract

What to do

Build interaction machine of S contract.

What is supposed to be clear before building an interaction machine of S contract

- The possible states of the simple system when S receives a specific request and when S issues the responses.
- The possible activities in the interactions of S.
- The conditions for the occurrence of each interaction, i.e the guards of the arc.
- The parameters exchange between the interactions of S.

The pattern of an interaction machine of S contract

- An initial state.
- The notes representing the possible interaction states of simple dynamic system representing S.
- The arcs representing the possible transitions of simple dynamic system representing S, labelled by an interaction occurrence, a guard and an effect in the form of *interact-occur [guard]*, where *interaction-occur* may be an input interaction and an output interaction that are defined in S interface.
- A number of final states according to a number of possible ends of S behaviors.

Building steps

1. Start the graph with an initial state
2. Build the arc for the first input interaction.
3. Define and name the next states of the simple dynamic system representing S when S receives the requests or processes the responses.
4. Define the attributes of the data of the input interaction and their possible values to build the boolean expressions for the guards of the following transitions.
5. Corresponding to the specific guards, build the next arcs for the following transitions.
6. Mark the points where the behaviours end in final states.

Note:

- An interaction machine may have any number of final states.
- Whenever there is an arc with a condition there should be also the arc with opposite condition.
- If no condition is satisfied for an elementary interaction, the matching interaction will never be matched.

Give the simple dynamic type representing the realm of S
Model interaction machine of a semantics

What is supposed to be clear before building an interaction machine in a semantics

- The service contract
- The domain of the value of the simple dynamic system attributes.
- The effect of the activities of the simple dynamic system.

The pattern of an interaction machine in a semantics

- An initial state.
- The notes represent the possible interaction states of the system.
- The arcs represent the possible transitions of the system, labelled by an interaction occurrence, a guard and an effect in the form of *interact-occur [guard] / effect*, where *interaction-occur* may be a input interaction and a output interaction that are defined in the provided service interface, the *[guard]* is the boolean expression built over the simple dynamic system attributes, and the *effect* is the action over those attributes.
- Final states (also none).

Building steps an interaction machine in a semantics
On the basis of the provided interaction machine of the service contract,

1. Create the arcs to define the initial value for the simple dynamic system attributes if any.
2. Supplement the possible value of the attributes of the simple dynamic system to create the *guard* for the transitions.
3. Define the effect of the interactions over the simple dynamic system attributes to create the *effect* for the transitions if any.

Note:

- The main difference between the service contract and the semantics is the presence of the values of the Simple system attributes in the *effect* of the transitions.

Model participant P

A-Model monolithic participant P

What to do

Define the monolithic participant P and identify the services that P provides and consumes

Building steps

- Use CASL-MDL simple dynamic type with stereotype ≪Participant≫ to define P, name this simple system with the name of P.
- Create a port with a lollipop for each service that P provides, name this port by the service name.
- Create a port with a cup for each service that P consumes, name this port by the service name starting with conjunction ∼.

B-Model structured participant P

What to do

Identify all subparticipants of P and the local services that they provides and consumes.

Building steps

- Use a structured system (i.e., CASL-MDL structured dynamic type) with stereotype ≪Participant≫ to define P, name this structured system by the name of P.
- Use a subsystem to define each subparticipant. Insert them into the structured system P.
- For each subparticipant, create a port for a service that it offers or uses.
- Connect the pair of participants involved in a service by a connector labelled by the name the service.
- Create a port for participant P for a service that it provides (a port having a lollipop) or consumes (a port having a cup). Connect this port to the corresponding port of subparticipant which provides or consumes this service.

Model structured participant P as service architecture of the service system

A service architecture is represented by a CASL-MDL structured system of a number of subsystems.

What is supposed to be clear before building a service architecture:

- All the participant types of the service system.
- All the services of the system.

The composition of the service architecture

A structured system stereotyped ≪serviceArchitecture≫ includes:

– The participant types are represented by subsystems, on which the ports are the points they offer or use the service.
– The connectors between the participants.

Building steps for a service architecture:

– Use a structured system (i.e., CASL-MDL structured dynamic type) with stereotype ≪Service Architecture≫ to define P, name this structured system by the name of service system.
– Use a subsystem to define each subparticipant. Insert them into the structured system P.
– For each subparticipant, create a port for a service that it offers or uses.
– Connect the pair of participants involved in a service by a connector labelled by the name the service.

Note:

– The connector leaves and ends at the ports of the participants.
– The interactions occurring among the participants are in terms of services.

Model service S in property oriented style

A-Build the contract of service S in property oriented style

The contract of S in property oriented style characterized by formulas expresses all possible occurrences of the interactions in the particular conditions.

What is supposed to be clear before building the contract of service S in property oriented style

– The protocols to be followed by the providers and the consumers of the service, what they guarantee to each other and what they expect in terms of dynamic behaviour.
– What will happen when the service receives a specific request.
– What conditions match each interaction and how they match with each other, such as what conditions for a request to be accepted, or what conditions for a response be implemented.
– The attributes of the data in a request sent to the service by the consumers, what is true or not.
– The relationships between the interactions (the order, the dependency and the priority)

The composition of S contract in property oriented style

– A set of formulas expresses all possible cases of the occurrence of the interactions defined in the provided service interface according to the specific matching conditions.

Building steps

1. Define the context of the transition of the Simple system to select the quantifications on paths and quantification on states if any.

2. Define the possible values of the parameters and the attributes of the data of the specific input interaction, combine them with this input interaction to build the **premise** of the formulas.
3. Based on the relationships between the conditions and the interactions, between the input interactions and the output interactions, use the logic combinators and given quantifications to build the **conclusion** of the formula, such that each formula may expresses all possible interactions of the system corresponding to each specific premise part.

B-Model the semantics of service S in property oriented style

The semantics of S in property oriented style not only expresses all possible occurrences of the interactions in the particular conditions but also the possible internal transitions associating with the effect of the transitions if any.

What is supposed to be clear before building the semantics of S in property oriented style

- What contributes the answer of the service.
- What is examined, controlled, affected by the service itself.
- The provided attributes of the simple system that can determine its internal states, the internal actions over those attributes.
- The effect of the request sent to the service itself.
- The effect of the response to the service itself.

The content of S semantics in property oriented style

- A set of formulas expresses all possible cases of the occurrence of the interactions defined in the provided service interface according to the specific matching conditions; express the internal states of the simple system and the internal actions over the simple system attributes associating with such interactions.

Building steps

1. On the basic of the premise parts of the provided formulas in the service contract in property oriented style, combine them with the possible properties of the attributes of the simple system to create premise parts for the formulas.
2. Corresponding to the premise parts, create the the conclusion parts of the formula for the occurrences of the interactions and supplement the effect of the transitions if any.
3. Create necessary formulas to define the conditions for the simple system attributes to guarantee the realization of the transitions.

Note: The effect of the transition is usually attached with the corresponding interaction by logic combinator AND.

12 Dealer Network Model in Casl4Soa

In this section, we model the Dealer Network using CASL4SOA, first following the constructive style and later following the property-oriented one. The description of Dealer Network case study has been given in Sect. 2.1.

The Dealer Network is a service system, and thus is a special structured participant without ports for offering and consuming services. The Dealer Network CASL4SOA model, in both styles, is thus a special case of participant model consisting of:

- the definition of a dynamic type stereotyped by ≪ServiceSystem≫ corresponding to the Dealer Network and named DealerNetwork, by means of a type diagram including such type; DealerNetwork is a CASL-MDL structured dynamic type (thus the definition of this type will also express the service architecture of the Dealer Network);
- the models of the local services, that are Place Order, Get Ship Status, and Request Shipping;
- the models of (the types of) its subparticipants, that are Dealer, Manufacturer, and Shipper.

12.1 Dealer Network Model in Casl4Soa: constructive style

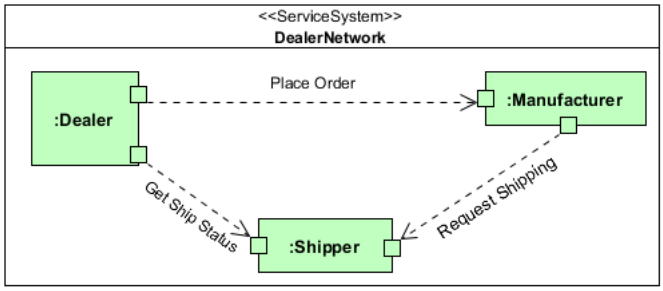

Fig. 42. DealerNetwork type

DealerNetwork type DealerNetwork, the dynamic type for the service system corresponding to the Dealer Network, is shown in Fig. 42 by a box (the dynamic type icon of CASL-MDL) stereotyped by ≪ServiceSystem≫ to express that it is a service system. Recall that in CASL4SOA, we denote the participants of a service system (as well as the subparticipants of a generic structured participant) by subsystems in the structured system stereotyped by ≪ServiceSystem≫ (stereotyped by ≪Participant≫). Each subsystem (depicted by a box) represents a role for (sub)participants of a specific type (the type name is depicted in the box

after the colon). Moreover, the fact that a (sub)participant interact with another (sub)participant by means of a service is shown by a dashed arrow going from the port of who uses the service towards the port of who provides it. Thus Fig. 42 shows also the service architecture of Dealer Network in CASL4SOA. We can see that there are several participants of three different types Dealer, Manufacturer, and Shipper. Each participant has a number of ports to indicate that it provides or consumes various services. For example, the participants typed by Shipper have two ports for the two services that they provide, they are Get Ship Status and Request Shipping. The dashed arrows entering in the Shipper ports are labelled by the names of the services that it provides.

Local Service Models

Service Place Order

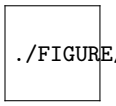

Fig. 43. Place Order Service: interface
1

Service Interface The interface of service Place Order, shown in Fig. 43, is a type diagram that contains a dynamic type named Place Order with stereotype ≪serviceInterface≫ modelling the the service interface itself, and the definitions of the datatypes needed to type the parameters of its interactions.

The interface of service Place Order consists of four interactions: ?_request_Quote and ?_place_Order of kind input, and !_quote and !_order_Status of kind output. The service can receive the quote request from the buyers by interaction ?_request_Quote with a parameter typed by the datatype QuoteRequest, then the service may respond with the quote contained in a parameter typed by datatype Quote of the output interaction !_quote. When the service receives the request to place an order from the buyer by the interaction ?_place_Order, it will respond by communicating the status of the order by means of the interaction !_order_Status. The datatype OrderStatus contains a boolean attribute confirmed to record the fact that the order has been confirmed or not. Moreover, if the order is confirmed, the buyer will receives further information about the order contained in the other attributes of OrderStatus, they are the providerID of the order, the delivery date of the shipment, and the wBN (waybill number) of the shipment. The identification of the buyer is defined by the attribute customerID in the definition of datatype QuoteRequest and Order.

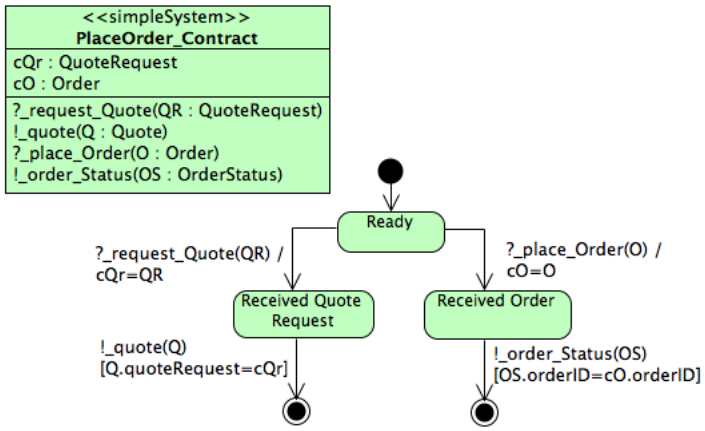

Fig. 44. Place Order service: contract (constructive style)

Service Contract The contract of service Place Order is represented by the simple dynamic system PlaceOrder_Contract and the interaction machine shown in Fig. 44; this dynamic system has the same interactions of the one modelling the service interface, and all of them will appear on transitions between of this interaction machine. The interaction machine expresses that the service may receive two requests: the quote request and the order placement from the consumers, through the two transitions leaving the state Ready labelled respectively by ?_request_Quote(qr:QuoteRequest) and ?_place_Order(O). The attribute cQr of the simple dynamic system allows to store the value of the parameter quote request QR; similarly, the attribute cO stores the order O. Those values are used in the transition guards to determine the cases when the interactions may happen. For instance, to guarantee that the quote of each quote request will correspond to such request, the guard for the output interaction !_quote(Q) should be [Q.QuoteRequest=cQr].

Semantic View The semantics of service Place Order is modelled by the simple dynamic system PlaceOrder_Semantics and the interaction machine shown in Fig. 45. In this example, the interaction machine of the semantics has a similar shape to the one of the service contract (see Fig. 44), however it takes also into account the information about the realm. In this case, the quantity of product in stock is this information, thus we introduce attribute stock in system PlaceOrder_Semantics. The interaction machine in Fig. 45 models that, if the ordered quantity is greater than the product quantity in stock, the order will be cancelled; otherwise the order will be confirmed, and the product quantity in stock will be reduced by the ordered quantity.

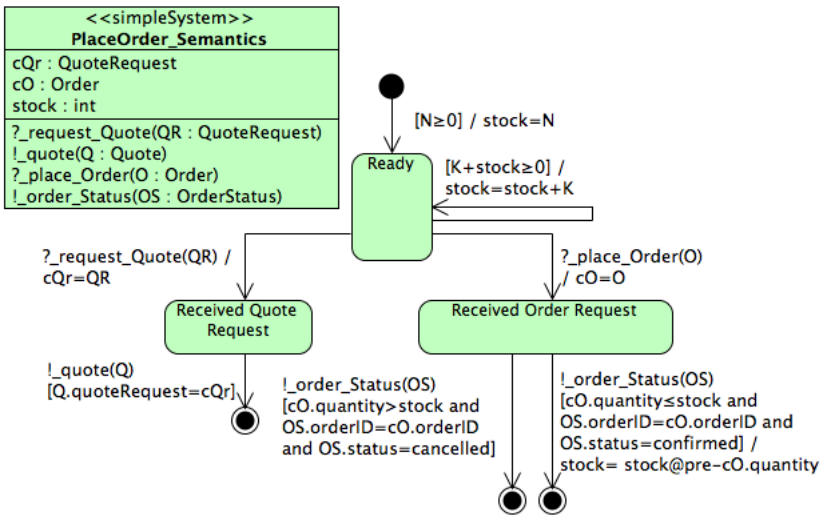

Fig. 45. Place Order service: semantics (constructive style)

Service Request Shipping Fig. 46 shows the interface of the service Request Shipping. The service provides a means to require a shipping request by ?_request_Shipping(R:Request) and it can respond the two possible results, one for package packing and another for confirmation of delivery, in !_package_PickUp(PP:PackagePickUp) and !_delivery_Confirmation(DC:Confirmation).

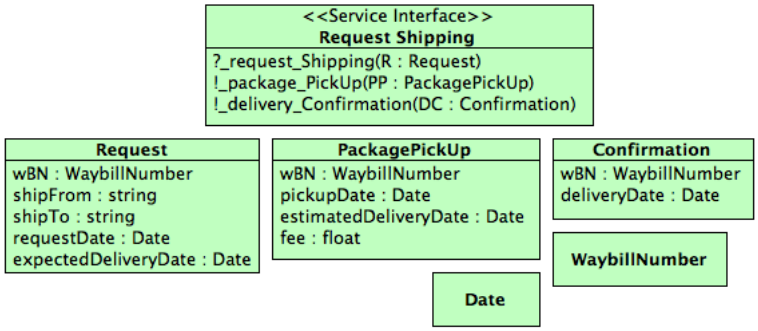

Fig. 46. Request Shipping service: interface

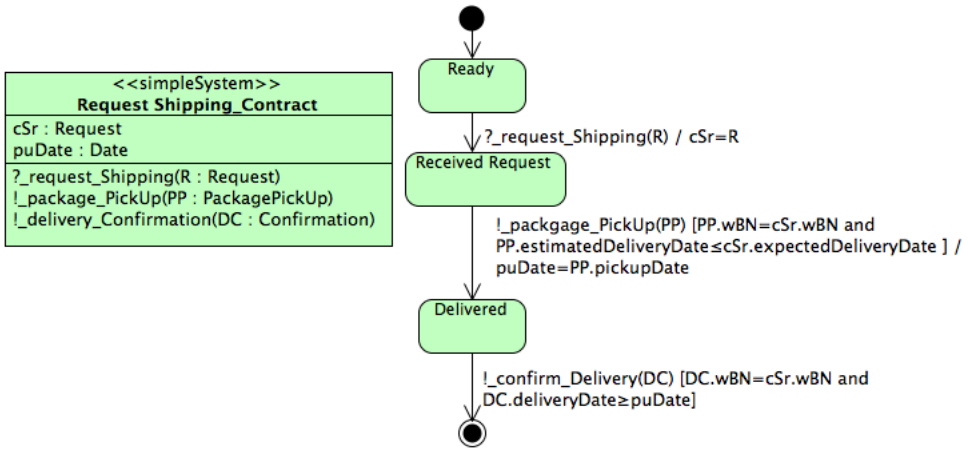

Fig. 47. Request Shipping service: contract (constructive style)

Service Get Ship Status Fig. 48 shows the interface of the service Get Ship Status. The service provides a means to require the status of a shipment by message ?_get_ShipmentStatus(W:WaybillNumber) and it can respond the result in !_shipmentStatus(SS:ShipmentStatus).

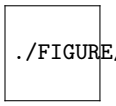

Fig. 48. Get Ship Status service: interface

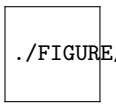

Fig. 49. Get Ship Status service: contract (constructive style)

The semantics is not very interesting and we do not describe the realm of the service, since that should include all the situations of the ships and of the sea and of the harbors, etc., so we should not present it.

Participant models The models of the three types of participants of Dealer Network (Dealer, Manufacturer, and Shipper) are shown in Fig. 50, to be more precise in such figure we show only the three type diagrams defining the three corresponding dynamic types stereotyped by ≪Participant≫, while for simplicity we do not duplicate the models of the services that they offer and use, since they have been already presented in the part about the local services of Dealer Network. We do not add any other information on these three participants, since we do not know anything other on them (e.g., if they are monolithic or structured, and what is their behaviour). CASL4SOA allows also these kind of specifications

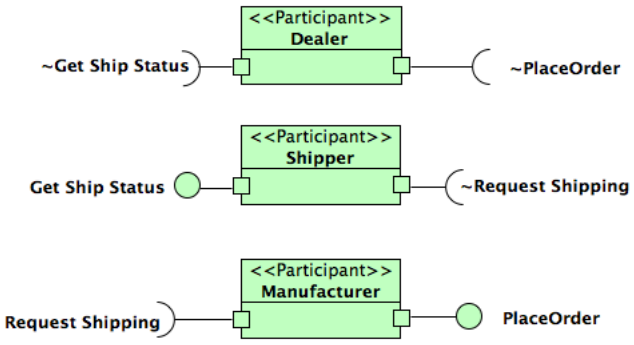

Fig. 50. Dealer Network service system: participant models

12.2 Dealer Network Model in Casl4Soa: property-oriented style

The CASL4SOA Dealer Network model made following the property-oriented style has the same structure of the constructive one presented in subsection 12.1, the only different parts are the definitions of the contracts of the three services (Place Order, Get Ship Status and Request Shipping) and of the semantics of Place Order. In this case they are defined by means of sets of constraints, i.e., set of temporal logic formulas. We present these contracts and this semantics in Fig. 51, 53, 54, and 52 respectively.

Fig. 51 presents the contract of service Place Order in property oriented style. The first formula expresses that whenever (in_any_case) the service receives a quote request, it will always (always) respond with a corresponding quote. The second formula expresses that whenever the service receives an order request, it will always respond with a corresponding order status which includes the boolean attribute confirmed. The correspondence is guaranteed by Q.quoteRequest=QR.

In the property-oriented style, the semantics of service Place Order is represented by a set of formulas (see Fig. 52). The first formula is the same as the

```
in_any_case always
    ?_request_Quote(QR) ⇒
        ∃Q:Quote • (Q.quoteRequest=QR ∧ eventually !_quote(Q))

in_any_case always
    ?_place_Order(O) ⇒
        ∃OS:OrderStatus • (OS.orderID=O.orderID ∧eventually !_order_Status(OS))
```

Fig. 51. Place Order service: contract (property-oriented style)

```
in_any_case always
    ?_request_Quote(QR) ⇒ ∃ Q:Quote • (Q.quoteRequest=QR ∧ eventually !_quote(Q))

in_any_case always
    ( ?_place_Order(O) ∧ O.quantity≤ stock) ⇒
        ∃ OS:OrderStatus • (OS.orderID=O.orderID ∧ OS.confirmed ∧
            eventually (!_order_Status(OS) ∧ stock=stock@pre-O.quantity))

in_any_case always
    ?_place_Order(O) ∧ O.quantity>stock ⇒
        ∃ OS:OrderStatus • (OS.orderID=O.orderID ∧ not OS.confirmed ∧
            eventually !_order_Status(OS))

in_any_case always
    ∃ K:int • (K+ stock≥ 0 ∧ eventually stock = stock@pre +K)
```

Fig. 52. Place Order service: semantics (property-oriented style)

first one in the service contract (see Fig. 51), because the quotation does not depend on the service realm. The second formula expresses that if the ordered quantity is less than or equal to stock, then the order will be confirmed, and an order status with attribute confirmed equal to true will be sent to the buyer; stock=stock@pre-O.quantity expresses that the stock will be reduced. Otherwise, as expressed by the third formula, if the ordered quantity is greater than stock, the order will be cancelled, and an order status with attribute confirmed equal to false will be sent to the buyer. The last formula states that the stock can always be modified adding or removing goods.

Fig. 54 shows the contract in property style of service Get Ship Status. It contains formula expressing that whenever the service receives a request for status of a shipment, it will respond the information that exists.

```
in_any_case always
    ?_request_Shipping(R) ⇒
        (eventually ∃ PP:PackagePickUp •
            (PP.wBN=R.wBN ∧
             PP.estimatedDeliveryDate ≤ R.expectedDeliveryDate ∧
             !_package_PickUp(PP))
             ∧
             eventually ∃ DC:DeliveryConfirmation •
                 (DC.wBN=R.wBN ∧
                  DC.deliveryDate ≥ PP.pickupDate ∧ !_confirm_Delivery(DC))
         )
```

Fig. 53. Request Shipping service: contract (property oriented style)

```
in_any_case always
    ?_get_ShipmentStatus(W) ⇒
        eventually ∃ SS:ShipmentStatus • (SS.wBN=W ∧ !_shipment_Status(SS))
```

Fig. 54. Get Ship Status service: contract (property oriented style)

13 Casl4Soa Model of Office System

In this section, we model the Office System using CASL4SOA, first following the constructive style and later following the property-oriented one. The description of the Office System case study has been given in Sect. 2.2.

The Office System is a service system, and thus is a special structured participant without ports for offering and consuming services. The Office System CASL4SOA model, in both styles, is thus a participant model consisting of:

- the definition of a dynamic type stereotyped by ≪ServiceSystem≫ corresponding to the Office System and named OfficeSystem, by means of a type diagram including such type; OfficeSystem is a special case of a CASL-MDL structured dynamic type (thus the definition of this type will also express the service architecture of the Office System);
- the models of the local services, that are Print, Check Italian, Check French, Check English and Publish on Web;
- the models of (the types of) its subparticipants, that are PrintingCenter, EnglishCenter, ItalianCenter, FrenchCenter, WebPublisher, and OfficeComponent.

13.1 Casl4Soa constructive model of Office System

OfficeSystem The type OfficeSystem is defined by the type diagram shown in Fig. 55. It is a structured system that is made participants of six different types providing and consuming five different services. The definition of the OfficeSystem type gives also the architecture of the service system OfficeSystem, showing which (roles for) participants of the various types use which services provided by which

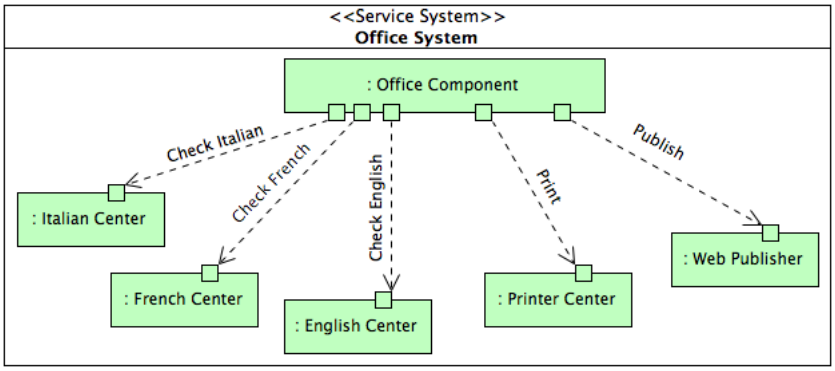

Fig. 55. Office System: service architecture

other (roles of) participant types. The fact that a participant uses a service provided by another one is shown by means of a dashed arrow going from the user to the provider and labelled by the name of the service. For example, we can see that OfficeComponent uses the Print service provided by the PrintingCenter, also it is clear that in this service system the participants typed by OfficeComponent uses the services provided by the participants of all the other types, and that the latter do not interact among them.

Local Service Model

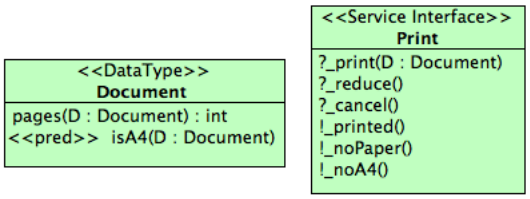

Fig. 56. Print service interface

Service Print The interface of service Print is shown in Fig. 56. Notice that also the datatypes defining the parameters of the in and out messages are defined in this type diagram, for example Document defines the printable documents; it contains an operation pages and a predicate isA4 returning respectively the number of pages of the document and the indication if the size of its pages is equal to A4.

Fig. 13.1 presents the contract of the service Print by means of a simple system and of an associated interaction machine. The interaction of this simple system are exactly the interactions appearing in the interface of the service define din Fig. 56, which correspond to the messages received and sent by the service; all of them appear at least on a transition of the interaction machine.

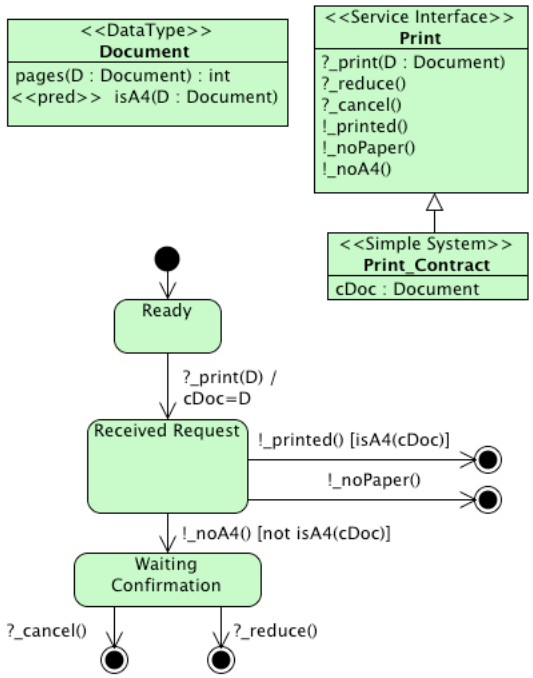

Fig. 57. Print service contract (constructive style)

The realm of the service Print is described by the paper quantity available, modelled by the attribute paper in the simple system Print_Semantics in Fig. 58.

The semantics of the service Print shown in Fig. 58 is defined by means of a simple dynamic type and an associated interaction machine. We can see now that the reason for getting the message !_noPaper; moreover, if we get the message !_noA4 we know that there is however enough paper to print the document (we can have a different service that after having received the message reduce may inform you that there is not enough paper to print). Moreover, we also know that each time a document is printed, the paper quantity will be decreased exactly by the number of its pages, and this is modelled by the effect / paper = paper - pages(cDoc) of the corresponding transition (again a different service may consume an extra page printing a forefront with the information on the data and the time of the printing).

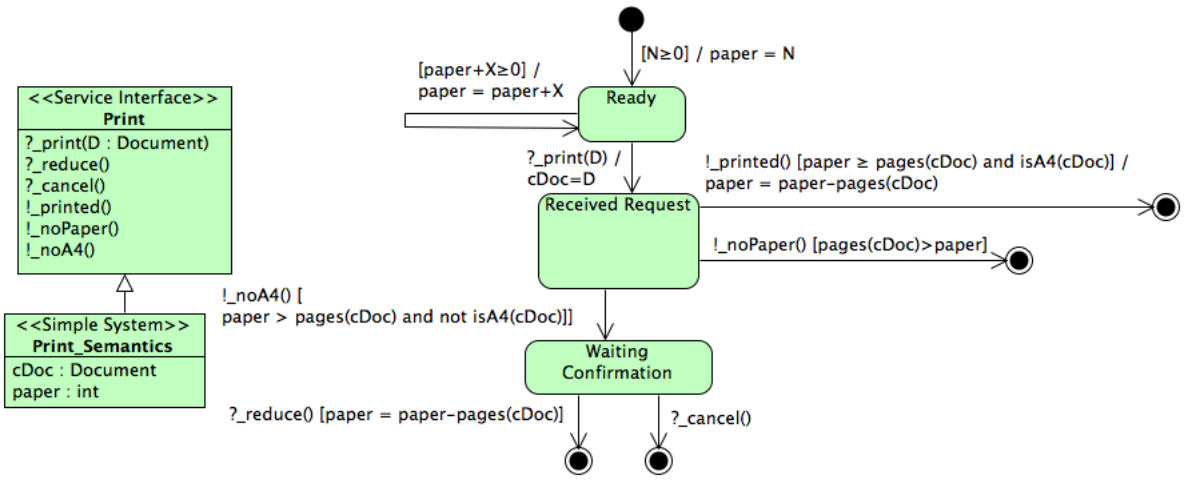

Fig. 58. Print service semantics (constructive style)

Services Check Italian Here we consider only the service Check Italian, the models of Check French and of Check English are similar.

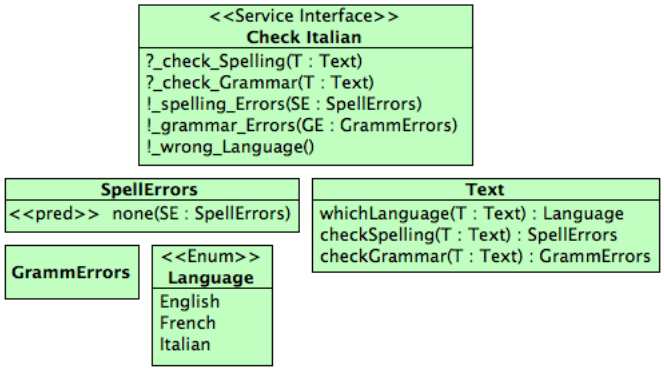

Fig. 59. Check Italian interface

Fig. 59 shows the interfaces of service Check Italian. The service offers two types of checking: the *spelling* and the *grammar* checking, that may be required by using the two input interactions of the service: ?_check_spelling and ?_check_Grammar. The content and the structure of the text is defined by the datatype Text, the spelling and grammar errors by the datatypes SpellErrors, and GrammErrors. The language of a text is detected by the operation whichLanguage of

Text. The return value of this operation can be one of the three values: English, French and Italian, which are listed in the enumeration type Language.

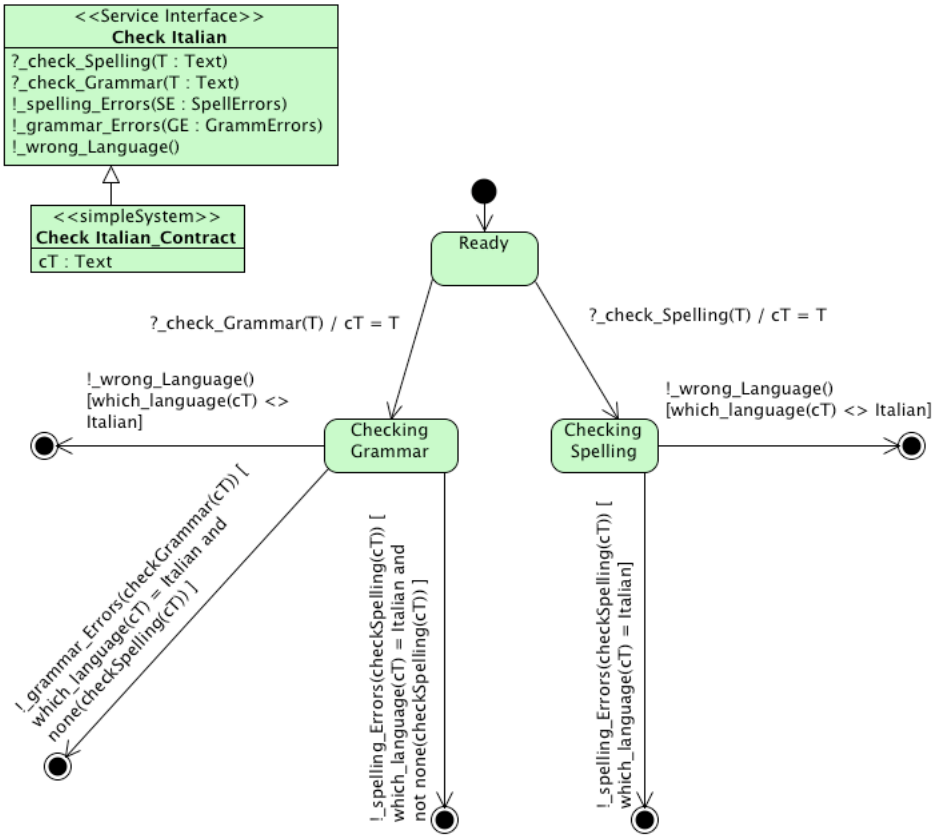

Fig. 60. Check Italian contract (Constructive Style)

In the service contract in Fig. 60, there are five final states illustrating the five results provided by service Check Italian in all possible cases. Before to check the spelling or the grammar, the service will check if the submitted text is written in Italian. If not, the service will send the message !_wrongLanguage. If the service finds some errors in the text, it will return the list of the found errors. If there is no error, this list will be empty. If there are spelling errors in a text required to be grammar checked, the service will return the spelling errors. It means that the service performs the grammar check if only if there are no spelling errors in the text.

We do not give the semantics of service Check Italian, since this service does not depend on a realm: we assume that the correctness of the spelling and of a language does not depend on any changeable aspects of the real world, and

moreover this service does not modify anything. If it is relevant to precisely specify how the spelling and grammar checks are made, it is possible to enrich the definition of the datatype Text by operation definitions or by constraints.

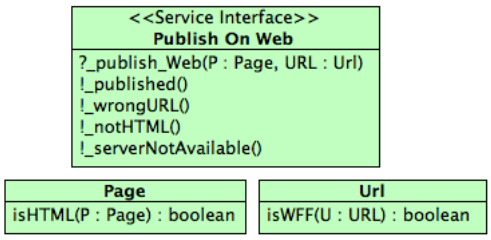

Fig. 61. Publish on Web service interface

Service Publish on Web Fig. 61 presents the interface of the service Publish on Web, whereas its contract is shown in Fig. 62. When a page published on the web, then the following conditions are satisfied: the page is in HTML format, the URL that the user provides is correct and its server is available at this time. In order to check whether the page is written in HTML and the provided URL is correct or not, two operations isHTML(P) and isWFF(U) are defined in the datatypes Page and URL respectively.

If the page is not in HTML, the service will communicate !_notHTML(), whereas if it is in HTML but the URL is ill-formed the service will send another error message. Finally, if both the document is in HTML and the URL is correct, the service may still send the error message about the server of the URL being not available.

The semantic view of service Publish on Web concerns the availability of the Web servers, and thus this is its realm, and it is modelled by the datatype Web The operations up and down modify the web status making a server available and not available respectively, whereas the predicate on checks if a server associated with a url is available.

Participants Six participant types of Office System are collected in the type diagram in Fig.64. They are represented by six ≪Participant≫ classes having ports typed by interfaces of the services they provide and consume. ≪Participant≫ Office Component is the participant type which only consumes services, thus all its ports are typed interfaces shown by the "cup" notation. Meanwhile, other participant types are providing participants which their ports are typed interfaces shown by the "lollipop" notation, for instance, ≪Participant≫ Printer Center has got the port Print representing for service Print that it provides.

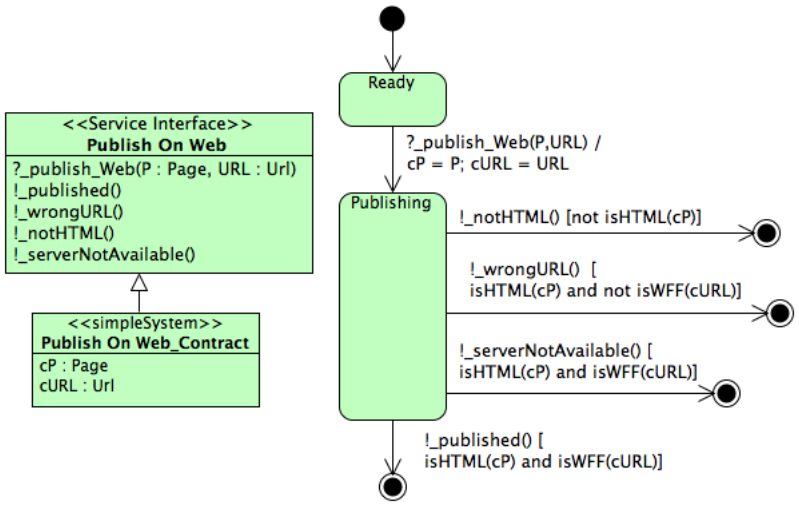

Fig. 62. Publish on Web service: contract (constructive style)

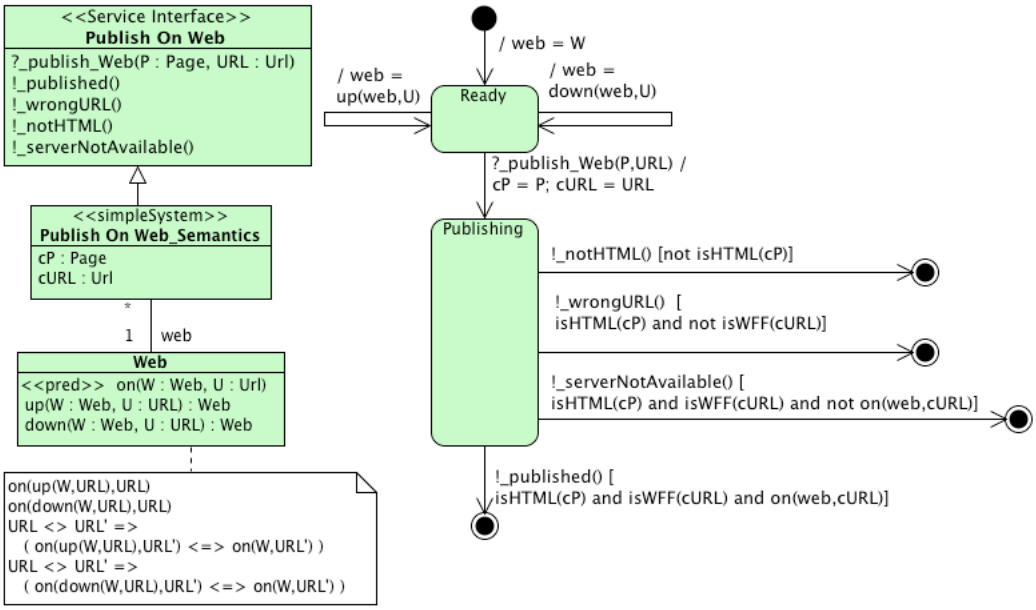

Fig. 63. Publish on Web service semantics (constructive style)

13.2 Property-oriented Casl4Soa Model of Office System

In this section, we model Office System using CASL4SOA in a property-oriented way, giving only the contracts and the semantics of the different services, since all the other parts are the same as in the constructive mode in Sect. 13.1.

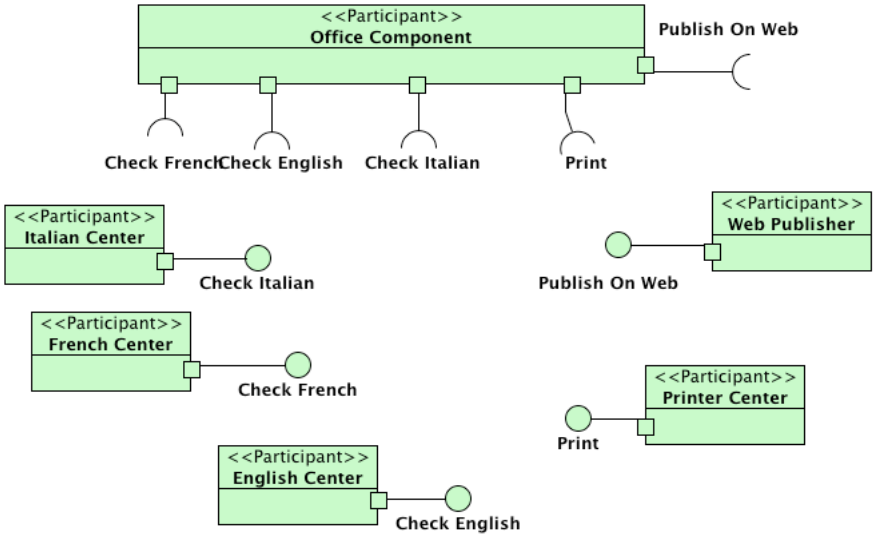

Fig. 64. Office System participants models

in_any_case always
(isA4(D) ∧ pages(D) ≤ paper ∧ ?_print(D)) ⇒
eventually (!_printed() ∧
(paper = N ⇒ next paper = N-pages(D))
in_any_case always
(not isA4(D) ∧ pages(D) ≤ paper and ?_print(D)) ⇒
eventually (!_noA4() ∧
eventually ((?_reduce()∧ (paper = N ⇒ next paper=N-pages(D)))
∨ ?_cancel()))
in_any_case always
(pages(D) ¿ paper ∧ ?_print(D)) ⇒ eventually !_noPaper()
in_any_case always
∃ X.X¿0 ⇒ eventually paper=paper+X

Fig. 65. Print service: semantics (property oriented style)

The semantics in property oriented style of service Print is characterized by formulas that are more precise than those in the contract. They describe not only the possible transitions of the system but also what justifies the answer, what is true or not, and the effects of the requests. For example, in Fig. 65, the second formula expresses that whenever the service receives printing request ?_print(D), meanwhile the paper is not in A4 format not isA4(D) and the printer has enough paper (pages(D) ≤ paper), it will communicate that the paper is not in A4 !_noA4() and either cancel the printing ?_cancel() or receive the page reducing from the user ?_reduce(). If the user reduces the pages, the effect of this interaction is that

paper will be decreased paper = N - pages(D). The last formula is defined in order to guarantee that the printer always will be refilled with paper.

```
in_any_case always
        ?_check_Spelling(T) ∧ whichLanguage(T)¡¿Italian ⇒
                eventually !_wrongLanguage()
in_any_case always
        ?_check_Spelling(T) ∧ whichLanguage(T)=Italian ⇒
                eventually !_spelling_Errors(checkSpelling(T))

in_any_case always
        ?_check_Grammar(T) ∧ whichLanguage(T)¡¿Italian ⇒
                eventually !_wrongLanguage()

in_any_case always
        ?_check_Grammar(T) ∧ whichLanguage(T)=Italian ∧ none(checkSpelling(T)) ⇒
                eventually !_grammar_Errors(checkGrammar(T))

in_any_case always
        ?_check_Grammar(T) ∧ whichLanguage(T)=Italian ∧ not none(checkSpelling(T)) ⇒
                eventually !_spelling_Errors(checkSpelling(T))
```

Fig. 66. Check Italian: contract (property oriented style)

```
in_any_case always
        (isHTML(P) ∧ isWFF(U) ∧  ?_webPublish(P,U)) ⇒
                eventually  (!_published() ∨ !_serverNotAvailable())
in_any_case always
        (not isHTML(P) ∧ ?_webPublish(P,U)) ⇒
                eventually  (!_notHTML())
in_any_case always
        ( isHTML(P) ∧ not isWFF(U) ?_webPublish(P,U)) ⇒
                eventually  (!_wrongURL())
```

Fig. 67. Publish on Web service: contract (property oriented style)

The semantic view of service Publish on Web concerns the availability of the Web server. The datatype Web built in Fig. 63 can be also regarded as the Internet in general. The operation up(W,P) that returns the Web type expresses the available status of the Web server, otherwise the operation down(W,P) expresses an unavailable status. It is important to note that these statuses are temporal for an actual Web server at specific moment. Later on we shall use those operations to define the semantics in property oriented style for this service in Fig. 68.

on(up(W,U),U)
U ≠ U' ⇒
on(up(W,U),U')⇔ on(W,U')
not on(down(W,U),U)
U ≠U' ⇒
on(down(W,U),U')⇔ on(W,U')

Fig. 68. Publish on Web service: semantics (property oriented style)

Fig. 68 defines the properties of the Web server in two formulas that express the conditions for the Web server to be available. The Web server is available if and only if it will turn back to the status up after it was in status down before, while the URL can be different. It means that if the Web server is available, it will not maintain the down status for ever. Therefore at last the page will be published in this case.

References

1. Object Management Group. *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)*, May 2012.