On-the-fly Construction of Web Services Compositions from Natural Language Requests.

Alessio Bosca and Fulvio Corno
Politecnico di Torino, Torino, Italy
{alessio.bosca, fulvio.corno}@polito.it

Giuseppe Valetto
IBM T.J. Watson Research Center, Hawthorne, NY, USA and Telecom Italia Lab, Torino, Italy
gvaletto@us.ibm.com
giuseppe.valetto@tilab.com

Roberta Maglione
Telecom Italia Lab, Torino, Italy
roberta.maglione@tilab.com

***Abstract*— The introduction of Semantic Web techniques in Service-oriented Architectures enables explicit representation and reasoning about semantically rich descriptions of service operations. Those techniques hold promise for the automated discovery, selection, composition and binding of services. This paper describes an approach to derive formal specifications of Web Service compositions on the basis of the interpretation of informal user requests expressed in restricted Natural Language. Our approach leverages the semantic and ontological description of a portfolio of known service operations (called the Semantic Service Catalog). Each user request is processed against a Natural Language vocabulary that includes lexical constructs designed to convey the operations' semantics, in order to recognize and extract fundamental functional requirements implied by the request, and associate them to entries in the Catalog. In addition, the request interpreter extracts from the request the overall service logic, expressed in terms of a set of modular templates describing control and data flow among the selected operations. The result is a composition specification that associates on demand each user request to a new composed service. That specification is formal and can thus be transformed in an executable flow document for a target service composition engine.**

***Index Terms*—Service-Oriented Architectures, Semantic Web, Semantic Web Services, Natural Language Processing.**

I. INTRODUCTION

The recent introduction of Semantic Web [1] ideas and results in the field of service-oriented computing has originated a vision of *Semantic Web Services* [11, 12], predicated upon the machine understandability of the nature of operations made available as Web Services. The linguistic and ontological means for representing properties and the capabilities of Web Services, and thus enhancing the ability to reason about the tasks they perform, seem particularly appealing for the support, based on operation semantics, of highly dynamic service selection and composition, which is an important goal of the Web Services (WS) paradigm. A major outstanding challenge to reach that goal is how to map the requirements describing a complex, composite service-oriented application (sometimes called a Value-Added Service, or VAS) to a multiplicity of simple, atomic Web Service operations, as well as to an overall service logic that coordinates their interactions to reach the goals of the intended VAS.

Re-use is one of the key elements in the computing paradigm of Service Oriented Architectures (SOAs). Therefore, numerous works, such as [20, 23], investigate how to create a composite service as an aggregate of existing services, looking at those services as building blocks. The most widespread approach for service composition is probably the workflow-based one, in which the logic of the overall composition and the interactions among atomic services, as well as between services and the VAS users are formally expressed by means of a workflow language (also called sometimes a process specification language). In the workflow-based composition methods, we can distinguish between static and dynamic composition.

Static composition requires that an abstract (non-executable) process model is manually produced *a priori*, i.e., before the composition is invoked. The abstract process model includes a set of tasks and their data and control flow dependencies. Each task contains a query clause that is used to search a Web Service to fulfill that task. In that case, at most only the selection and binding of atomic Web Services is done automatically. EFlow, [21], for example, is a platform for the specification, enactment and management of composite services. A composite service is modelled by a graph that defines the

order of execution among the nodes in the process. Notice that once the graph is created manually, it can be later updated for each execution request.

On the other hand, dynamic composition automatically generates a workflow as well as selects atomic services to fulfil the tasks in that workflow. That is done at invocation time and requires the requester to specify enough characteristic and constraints of the requested VAS to guide the automated composition.

Some approaches for workflow-based composition leverage the use of workflow templates [24], on the basis that evolving requirements in a VAS, as well as the constantly changing availability of services makes it hard to build valuable fixed workflows. Workflow templates enable describing the abstract functionality that is needed, so concrete services that satisfy the requirements can be selected at run time. To describe abstract functionality and encode qualitative preferences in workflow descriptions, ontologies can be uses, which allow to accommodate both hard constraints about the functional parameters of the services and soft constraints related to non-functional attributes of the services.

McIlraith et. al. [25, 26, 27] adapt and extend the Golog language for dynamic construction of composed services. Golog is a logic programming language built on top of situation calculus. The authors address the Web service composition problem through the provision of high-level generic procedures and customizing constraints. Golog is adopted as a formalism for representing and reasoning about this problem.

Our work presents an approach for the automatic generation of a high-level VAS specification (*Abstract Composition* in the remainder) on demand, starting from informal user requests expressed in (restricted) Natural Language. Our approach leverages several components: a set of simple and modular *logic templates* for representing the overall control and data flow of the VAS; semantic, ontological information about the operations exposed by a portfolio of Web Services, and a dictionary of Natural Language constructs that can be mapped to items included in that portfolio. We employ OWL-S [2] annotations to provide a formal representation of the service and operation semantics and an ontological description and classification of the Web Services in our portfolio. Those annotations are also leveraged by the Natural Language dictionary during the interpretation of user requests, in order to derive intentions, and then extract the major functional requirements, constraints and other useful information about the required VAS. By coupling in that way ontological tools typical of the Semantic Web with Natural Language Processing (NLP) techniques, we obtain a sort of semantic glue that bridges human usability to machine understandability.

The Abstract Composition generated from the interpretation of a user request can be translated into an executable flow, and maps the user needs and intentions – as inferred from the original request - to known Web Service operations that can satisfy them, in a "task-oriented" way, as envisioned for example in [7].

This paper focuses primarily on how we couple semantic and ontological information with Natural Language constructs, in order to specify Web Services compositions, and empower on-demand VAS creation composition.

The paper is organized as follows: in Section II we present an overview of our approach, and how it is mapped onto a prototype system for on-demand service composition. In Section III, we describe in detail how we use OWL-S to construct a semantic catalog of Web Services; in Section IV we discuss NLP techniques we employ for request interpretation, and how they relate to the OWL-S semantic information discussed in Section III; in Section V, we present the resources comprised in the dictionary; Sections VI and VII provide details on the request interpretation process. Finally we conclude with a discussion of related work, in Section VIII, and an outlook on future research, in Section IX.

II. Approach Overview

Our approach for the specification of service compositions at run-time has two starting points: the user request, which is processed and interpreted on the fly, to elicit functional requirements as well as a high-level view of the composition logic implied in the request; and a repertoire of well-known services that are described by rich semantic meta-data. Those two elements are, respectively, the Request Interpreter and the Service Catalog, displayed in the dotted box of Figure 1, which also depicts other elements of our composition system.

Our technique follows from two major assumptions:

1. user requests are relatively simple and concise, in structure and terminology, and can be satisfactorily expressed with a controlled subset of natural language;
2. furthermore, a common ontological vocabulary can be established, and is consistently applied to all entries in the Service Catalog.

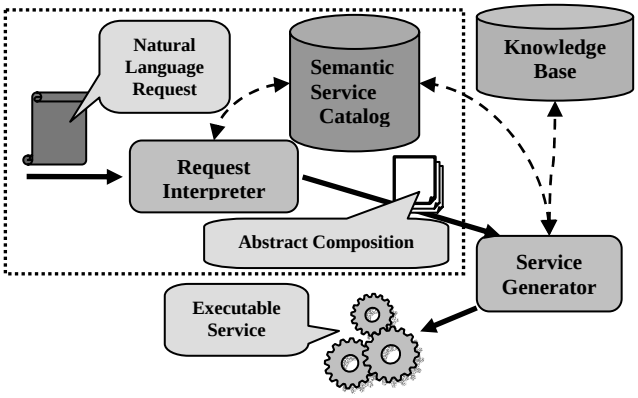

Figure 1. System overview.

While the latter assumption is probably unfeasible in a context, in which Web Services that are owned by multiple parties and are located anywhere over the Internet at large should be summoned in response to the user re-quest, it becomes reasonable and manageable in the context of a limited set of Web Ser-vices that are kept

under the control of a single entity, like in the case of a provider or operator that offers value-added services to its customer base.

About the first assumption above, the rationale is twofold. On the one hand, NLP in a general purpose environment is notoriously an extremely hard challenge; however, within a limited and well-known context, it is possible to significantly reduce reduce its inherent complexity. On the other hand, we have observed certain (and large) user communities are already accustomed these days to request "canned" or at best parameterized VAS through semi-formal, restricted text (consider, e.g., subscription services that can be activated by using text messaging and SMS on a user's cell phone); that kind of interaction is considerably more constrained than the kind of controlled Natural Language that we support, as made evident from the examples we provide further down, in Section VI.

Both the Request Interpreter and the Service Catalog in Figure 1 have as their foundation a common ontological framework expressed in OWL-S, which provides us with the means to semantically describe and categorize each atomic service operation. Those semantic descriptions are centered upon the OWL-S concept of service *Effect*. Besides being used for the annotation of Web Services included in the Catalog, we exploit OWL-S also in the Request Interpreter, to support NLP techniques. Starting from a user request expressed in Natural Language (NL) the Request Interpreter is in charge of decomposing the sentence in order to detect and isolate known Natural Language expressions that are associated to the Effects listed in the Semantic Service Catalog, and hence mapped onto specific service functionality provided by the corresponding atomic operations. At that end, the Request Interpreter translates the NL request into an Abstract Composition document, that is, a formal VAS specification. That specification includes a lattice of logic templates, describing how they relate to and can be composed with each other into the global VAS flow; moreover, it includes a list of Effects, which act as generic, semantic placeholders for operations that must be invoked along that flow.

Our approach also includes mechanisms to transform that abstract service specification into a concrete one. This transformation is carried out by a Service Generator engine. Functionally speaking, the role of the Service Generator - as shown in Figure 1 - is simply to translate Abstract Compositions into a notation that can be executed over a service-oriented runtime of choice (for example, an orchestration environment employing BPEL4WS [6, 3], or a choreography environment [13]). However its task is multifold and its structure is complex. For example, the Service Generator is in charge of turning the lattice of logic templates into a single flow document. Moreover, it uses the recognized Effects and the Service Catalog to retrieve any known service operations that satisfy those Effects (multiple alternative operations can exist in the Catalog for a given Effect). Additionally, the Service Generator accomplishes the binding of all service operations in the executable flow document to existing Web Service in-stances (as proposed for example in [15]) Finally, the Service Generator may need to go back to the user, to resolve outstanding ambiguities and get an acknowledgment before the final generation of the executable VAS.

Given the complexity of the Service Generator, a fuller, in-depth discussion of our solution, detailing its internal architecture, mechanisms and algorithms is not feasible here due to space limitations, and remains outside the scope of this paper.

III. A Semantic Service Catalog In OWL-S

OWL-S is a framework to describe services from several perspectives: more precisely it characterizes services through a set of sub-ontologies.

- The ServiceProfile contains information meant for service advertising and discovery.
- The ServiceModel describes how the service works; for instance, IOPEs (Inputs Outputs Preconditions and Effects) offer a semantically rich mean to describe services operations.
- The ServiceGrounding provides details for service invocation; OWL-S defines by default a WSDL-based grounding, but the framework can accommodate also other groundings.

We have analyzed OWL-S in order to establish which concepts are suitable to be specialized and/or extended, in order to build sufficiently expressive service descriptions for automated discovery and composition purposes. We have recognized the need for models and algorithms to select services on the basis of semantic annotations stored not only in their profile (as proposed for example in [3]), but also in their IOPEs (see [4, 5]). We also propose to exploit IOPEs as a means to drive composition.

We have first of all come up with a way to model Effects in OWL-S, in order to enable the selection of service operations that satisfy some user requests or needs. Our modeling approach promotes the description of Effects in terms of the computing task that is performed by each atomic operation exposed by each service in our Catalog. We classify all of those operations onto OWL-S AtomicProcess elements; when we interpret a request, we provide for the dynamic composition of those atomic operations by generating a direct acyclic graph (DAG), where such AtomicProcess act as nodes and the relations among their IOPEs establish arcs. At invocation time, we translate that abstract graph into an executable service, using OWL-S grounding to bind each AtomicProcess to a concrete operation. To this end, we have implemented an ad hoc ontology called Effects (see Figure 2) for the classification of all instances of the Effect concept included in our SSC.

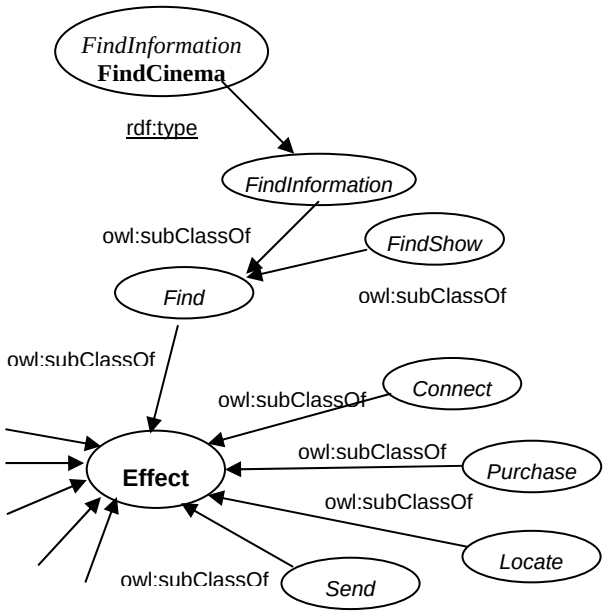

Figure 2. Effects Ontology.

Additionally, we focus on I/O parameters semantics referring to a set of concepts collected in another ad hoc ontology called IOtypes. In order to reason on inputs and outputs for the automatic, semantic-based composition of operations, we extended the OWL-S model with a couple of bidirectional properties that allow us to link processes to their I/O parameters and parameter to processes that can produce or consume them (see Figure 3).

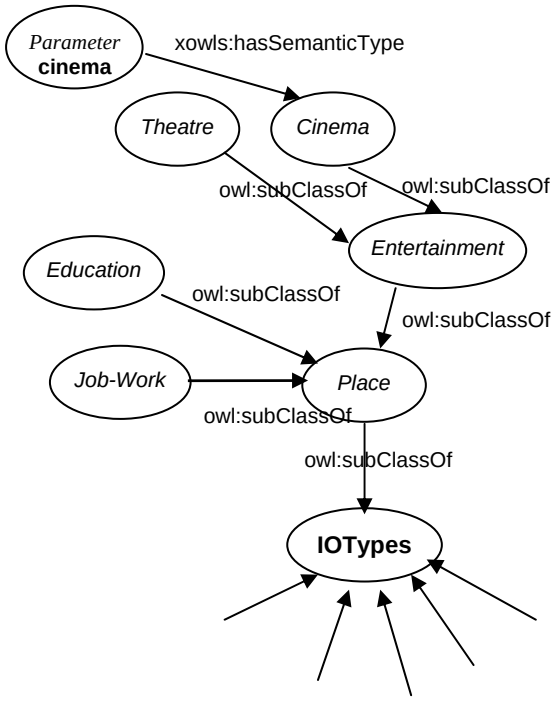

Figure 3. IOTypes Ontology.

As an example of our ontological description of AtomicProcess parameters consider the operation String[] findCinema(String title, String city), offered by a WS FindFilm included in our SSC. We map the operation findCinema to an AtomicProcess whose Effect refers to concept FindCinema defined in our Effects ontology, where FindCinema is an individual belonging to class FindInformation and is related to individual FindShow; we model findCinema inputs and outputs referring to some concepts from IOtypes ontology, e.g. the parameter city is semantically described by a CityName, a sub-concept of branch Geography in the IOtypes ontology, and the output computed by findCinema is mapped to a parameter exposing as semantic type a concept that descends from branch Entertainment (see Figure 4).

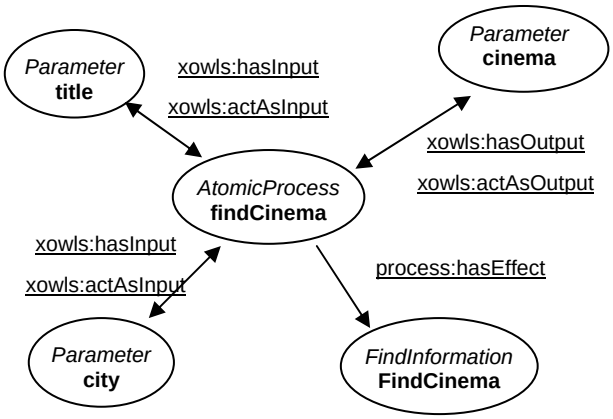

Figure 4. Find Cinema.

IV. Request Interpretation

Within the general architecture the Request Interpteter module is in charge of acquiring and processing textual user requests expressed in controlled natural language, and of translating them into a formal document (*AbstractComposition*) comprehensible to the service generation engine. Its main objective therefore consists in offering an interface for extracting functional requirements from those requests and mapping them onto a collection of well known services, that is, those described in our SSC.

Our approach is primarily built upon a dictionary of Natural Language constructs consistent with the service catalogue. Such dictionary consists of lexical entities associated with distinct elements of the catalogue and constitutes the main resource for the processing operations along with OWL-S services description itself.

Furthermore, the Request Interpreter accepts users' requests that are simple and restricted both in terms of sentence structure and lexicon. The sentence structure needs to be matched to the set of supported logic templates (for example, *if-then*, *while-do*, *sequence*, *conjunction*, etc.); the language must only include those lexical constructs that that are codified within the Natural Language dictionary.

Within the above mentioned constraints, - rather than completely parsing the request and try to fully analyze and interpret each single element, - it is possible to chose a simpler algorithmic approach that in a first step decomposes the request into fragments according to a proper logic template. Then it leverages the dictionary in

order to search for lexical patterns within each of those fragments and consequently infer the user's intention and eligible parameters. The figure below represents an overview of the operation flow.

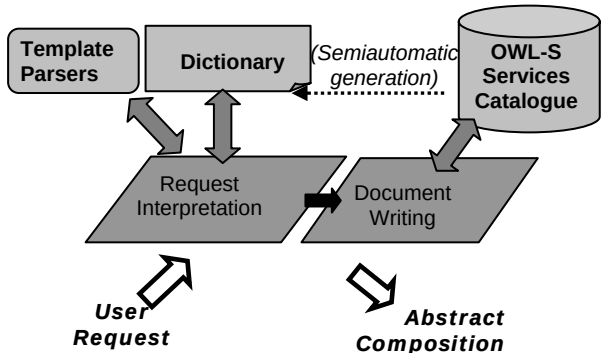

Figure 5. Operation Flow.

V. A TOOL FOR REQUEST INTERPRETATION: THE SSC DICTIONARY

In this section a brief summary is given on the resources comprised in the dictionary their features and how their contents are semi automatically derived from the SSC (Semantic Service Catalog), since that is a key point of our request interpretation strategy,. As stated before, the dictionary contains lexical elements related to some entities within OWL-S ontology and includes pure lexical resources (as lists of verbs or prepositions grouped by their role or meaning), as well as more complex ones, related to the sentence structure and its verbal governance (the *Sentence Constructions List* and the *Recognizer Catalog*).

The *Sentence Constructions List* is the main resource within the dictionary and it models the distinct expressive ways through which it is possible to request a service identified by a given *Effect* concept (see Figure 4 for the relation between OWL-S *AtomicProcesses* and *Effects*). We can in fact refer to a service by a functional request (*find a restaurant*, *reserve a ticket*...), or by a direct expression of user's intentions (*let me talk with*, *I want to see*...), or also by a statement about desired or unknown conditions concerning an object of user's interest (*in case it's raining*, *if the restaurant is open*...).

For each *Effect* present in the SSC a set of thematic keywords and a list of eligible constructions are explicitly listed in the docabulary, each construction specifying a group of verbs and a set of parameters. Every input or output parameter of any *AtomicProcess* related to an *Effect* is reported into such set as potential information that might be provided in the request by the user, given the intention modelled by the *Effect* itself. Each parameter is characterized by a certain *IOType* and by the role of the preceding preposition or by its position with respect to the verb. Output parameters, instead, constitute the values a user expects to be returned from a service or rather the conditions to be verified in a conditional logic block of the request. Effect individuals and their parameters are automatically extracted from the SSC and constitute the frame for populating the Sentence Constructions List for that Effect, while keywords, verbal classes, prepositions and other lexical entries have to be (in the current prototype) manually entered in the different slots. Figure 6 displays a fragment of the Sentence Construction List for the FindCinema Effect previously introduced.

```
<SENTENCE_CONSTRUCTIONS_LIST>
  <EFFECT ref="effect-c:FindCinema">
    <KEYWORD>film, cinema, movie, theatre, where
    </KEYWORD>
    <CONSTRUCTION id="1" verb="Find, Watch">
      <SAMPLE> Find out a cinema in –City-where –
      FilmName- is shown </SAMPLE>
      <IN param="parameter-c:filmName" type="io-
      types:FilmTitle" posWRTverb="Post"/>
      <IN param="parameter-c:city" type="io-types:City"
      prep="Where" />
    </CONSTRUCTION>
    <CONSTRUCTION id="2" verb="Show" >
      <SAMPLE> Where do they project –FilmName- in –
      City- </SAMPLE>
      <IN param="parameter-c:filmName" type="io-
      types:FilmTitle" posWRTverb="Post"/>
      <IN param="parameter-c:city" type="io-types:City"
      preposition="Where" />
      <OUT param="parameter-c:cinemaName" type="io-
      types:Cinema" prep="Where" />
    </CONSTRUCTION>
  </EFFECT>
  <EFFECT ref="effect-c:FindMeteoInformation">
    <KEYWORD>weather, forecast, sun, rain, wind, snow,
    temperature,.., </KEYWORD>
    <CONSTRUCTION id="1" verb="WeatherVerbs" >
      <SAMPLE>What's the weather like/Is it raining
      tomorrow in Rome?</SAMPLE>
      <OBJECT param="parameter-c:forecastDay"
      type="io-types:Date" positionWRTverb="ANY" />
      <OBJECT param="parameter-c:city" type="io-
      types:City" preposition="Where" />
    </CONSTRUCTION>
    <CONSTRUCTION id="2" verb="Tell, Know, Is" >
    <SAMPLE>Tell me the weather forecast for Saturday
    21 May in Turin</SAMPLE>
    <OBJECT param="parameter-c:forecastDay"
    type="io-types:Date" positionWRTverb="ANY" />
    <OBJECT param="parameter-c:city" type="io-
    types:City" preposition="Where" />
    </CONSTRUCTION>
    <CONSTRUCTION id="3" verb="Weather" >
    <SAMPLE>If tomorrow the temperature in Rome
    is..SAMPLE>
    <OBJECT param="parameter-c:forecastDay"
    type="io-types:Date" posWRTverb="ANY" />
    <OBJECT param="parameter-c:city" type="io-
    types:City" preposition="Where" />
    <RESULT param="parameter-c:forecast" type="io-
    types:Weather" posWRTverb="ANY" />
    <RESULT param="parameter-c:temperature"
    type="io-types:Temperature" posWRTverb="ANY" />
    </CONSTRUCTION>
  </EFFECT>
</SENTENCE_CONSTRUCTIONS_LIST>
```

Figure 6. Sentence Construction List.

The *Recognizer Catalog* is the other key resource within the dictionary and models the different information the system should be able to recognize as potential parameters. It focuses on the different *IOTypes* present in the SSC and specifies for each of them a set of features which enable the isolation and recognition of a given *IOType* within the text of a request. Such features both concern how the data appears and which value it holds; the recognition process in fact relies on data format, on the presence of a keyword or on the candidate parameter's occurrence within a given list. A set of *ad hoc* software modules can be associated to the various categories of input and output parameters identifies different basic information (as date, time, e-mails and telephone numbers) that recur in the dictionary, by means of lexical analyses or pattern matching with regular expressions. Information with the same format, but with different functional meaning is disambiguated through the presence of keywords, while proper nouns are confirmed by checking their presence in a proper thesaurus or a personalized list (such as the contacts in an address book).

Distinct *IOTypes* that bring the same functional information (as a city name or its latitude and longitude) are arranged into a compatibility group and can be found as valid values for the same parameters. Figure 7 reports a short excerpt of the dictionary, showing in bold the elements automatically retrieved from the SSC and in underlined italic the manually added lexical elements.

```
<RECOGNIZER_CATALOG>
  <IOTYPE id="io-types:BySipAddress"
  compatibilityGroup="io-types:Contact"
  format="SipAdress"> </IOTYPE>
  <IOTYPE id="io-types:GeographicalCoord"
  format="Numeric" > east, west, north,
  south, latitude, longitude </IOTYPE>
</RECOGNIZER_CATALOG>
```

Figure 7. Recognizer Catalog.

Notice that the language used for the prototype version of the architecture is Italian, although for a more general comprehension we have translated these examples in English for this paper.

VI. Request Interpretation Process

This section details the various phases of the interpretation process and describes how it exploits the lexical resources within the dictionary; the description is furthermore enriched by showing the results of the application of the procedure to the following sample sentence, whose contents have been chosen in order to fit in the scope of our SSC:

> *"If it rains tomorrow in Turin, reserve two tickets for "Sin City" at the Romano movie theatre".*

The first operative step consists in recognizing the logic flow underpinning the request and coupling it to one of the logic templates supported in the system: in this case, the correct logic template is the *if-then* template. A set of parsers properly tailored to the aforementioned templates process the request by trying to validate it against their own sentence model and if it matches, extract the distinct sentence blocks that refer to various parts of the template by tagging them as conditions or actions. The result of this phase for the example above thus consists in the identification of the *if-then* logical template and of two distinct clauses. If the sample sentence is fed into our system, the correspondent sub-output obtained can be summarized as follows:

> *<Template* ***id****='if-then'* ***condition****='it rains tomorrow in Turin'*
>
> ***action****='reserve two tickets for "Sin City" at the Romano movie theatre' />*

After this parsing procedure, we assume that different propositions have been identified and that each sentence block is constituted by only one clause with a principal verb and a set of objects. The following steps (2, 3 in Figure 8) consist in the interpretation of any individual action or condition retrieved in the previous phase.

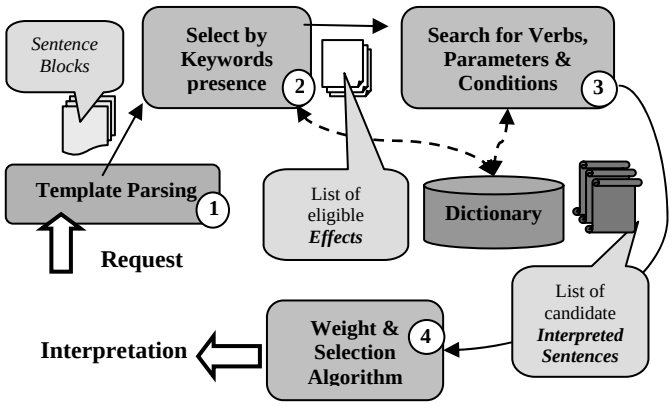

Figure 8. Interpretation Process.

The presence within the clause of a thematic keyword (recorded in the dictionary) provides hints about the user's intention and focuses the algorithm's attention on a set of services, considered as potential solutions and therefore inserted into a list of eligible *Effects*. In our sample sentence the presence of a term such as "*rain*" in the condition clause directs our attention to weather related services and terms as "*ticket*" or "*movie theatre*" in the action clause instead highlight services concerning entertainment and booking.

Our approach proceeds by guessing which information we have to look for, by taking in exam the hints contained in the *Sentence Constructions List*: if a suitable verbal form is found, a search is triggered over the sentence block for values that fit the parameters reported in the *Effect*'s construction. In our example, for what concerns the action block, we have to check the *Sentence Constructions List* of identified *Effects* that are associated

with the recognized keywords, In this example, a number of possible Effects apply: *FindCinema, FindCinemaShow, ReserveCinemaShow*[1] can all be selected on the basis of the lexical terms in the request, and search for text fragments that can be mapped onto the description of their parameters (cinema name, city, time schedule) will help disambiguate among them.

A proper recognizer is thus tuned according both to the functional features reported in the parameters' description (the position with respect to the verb or the introductive prepositions) and to the semantic ones reported in the correspondent *IOType* element of the dictionary. Consequently the recognition procedure tries to identify a block of text as eligible information.

If the functional information about where we expect to find the parameter (as for example after a given preposition or in a certain position with respect to the verb) and its "appearance" (the format of the data) are both verified as well as the semantic countercheck (the presence of a proper keyword in the same sentence chunk or the occurrence of the data in a list of known values), then a **strong** "match" is triggered. Otherwise, if the latter condition (the semantic countercheck) is not satisfied, a **weak** "match" is triggered.

Once all eligible parameter values are found, the textual fragments identified as weak matches are evaluated in order to be promoted or rejected; the promotion occurs if they have not been already assigned to another parameter slot, the rejection otherwise. Conditions, differently from other service parameters, can logically occur more than once in a sentence and form trees of elements logically related by means of AND or OR operators.

The two clauses, identified in consequence of the parsing by template (phase 1), are hence processed (phases 2 and 3) by the system; the logs stored during the algorithm execution are reported in Figures 9 and 10 as an insight to the operative steps. It is important to notice how the presence of the keyword *cinema* (on the action side) triggers the analysis of all the cinema related services present in our SSC and how a deeper investigation occurs only when a proper verb is found. Furthermore the example clarifies the promotion mechanism by showing, within the condition sentence, how the weak match of the *Turin* fragment as a city (no contextual information actually tells us whether such name is a city or not) results in a promotion because it has not been claimed by other parameter recognizers and so no other ambiguous interpretations are possible. Instead in the action clause the weak match of *Romano* as a city is rejected, because the textual fragment has already been identified as a cinema name with a strong match.

In such a case the city parameter recognizer fails and the final service description will result with an empty slot that should be eventually filled by the *Service Generator* (see Figure 1) by means of backward and forward composition techniques or by exploiting data stored in a user profile or by a direct interaction with the final user.

The analysis of each sentence block generates an *Interpreted Sentence* reporting the *Effect* id, the verbal form found, the list of the recognized parameters and the conditional expression, if present; the identified *Effect* models user intentions, under the assumption that they fit in the scope of the SSC, while the parameters represent the information pertinent to the service and retrieved within the request.

These *Interpreted Sentences* are gathered into a list of candidate solutions and processed by a ranking and selection algorithm (detailed in the following section) that constitutes the final step (phase 4) of the Request Interpretation. The algorithm works under the hypothesis that an interpretation holding more information and covering a larger part of the request should constitute a better solution.Therefore, the ranking assigns a score to the *Interpreted Sentences* for each element found (verbs, parameters, conditions) and then the interpretations with the highest rank are selected.

The structural information concerning the logical flow of the request, retrieved from the template parsing (phase 1), and the distinct *Interpreted Sentences*, obtained in the followings (phases 2-4), are then unified into an *AbstractComposition* document. The writing operation itself is accomplished as a further step in order to separate the actual interpretation from its concrete representation that is bound to a precise format. If more than one *InterpretedSentence* deserves the topmost rank and thus the selection algorithm is not able with the current approach to disambiguate them, then multiple alternative *AbstractCompositions* are provided to the *Service Generator.*

CONDITION>> *it rains tomorrow in Turin*

>>effect-c:FindMeteoInformation
-Construction ID 1
Weather **verb found** as *rains*
Weak Found of parameter-c:city with *Turin*
Strong Found of parameter-c:forecastDay with *tomorrow*
Promoted parameter-c:city with *Turin*
-Construction ID 3
Weather **verb found** as *rains*
Weak Found of parameter-c:city with *Turin*
Strong Found of parameter-c:forecastDay with *tomorrow*
Promoted parameter-c:city with *Turin*
Found simple condition parameter-c:forecast with *rains*

Figure 9. System Logs (Condition Block).

[1] Although the names are similar, the three Effects are functionally very distinct:, FindCinema is about locating a movie theatre; *FindCinemaShow* is about retrieving the schedule of a given movie; *ReserveCinemaShow* is about buying tickets and reseving seats for a given show.

ACTION>> *reserve two tickets for "Sin City" at the Romano movie theatre*

>>effect-c:FindCinema
-Construction ID 1
Strong Found of parameter-c:cinemaName with *Romano*
Weak Found of parameter-c:city with *Romano*

>>effect-c:ReserveCinemaShow
-Construction ID 1
Reserve **verb found** as *reserve*
Strong Found of parameter-c:tickets with *two*
Strong Found of parameter-c:cinemaName with *Romano*
Weak Found of parameter-c:filmTitle with "*Sin City*"
Weak Found of parameter-c:city with *Romano*
Promoted parameter-c:filmTitle with "*Sin City*"

>>effect-c:FindCinemaShow
-Construction ID 1
Strong Found of parameter-c:cinemaName with *Romano*
Weak Found of parameter-c:filmTitle with "*Sin City*"
Weak Found of parameter-c:city with *Romano*
Promoted parameter-c:filmTitle with "*Sin City*"

Figure 10. System Logs.

VII. Selection Algorithm

A weight is assigned to each *Interpreted Sentence* by the *Selection Algorithm*, according to the number and the types of the recognized parameters, the ratio between the recognized parameters, and the total parameters, also different weights are assigned depending whether matches are strong or weak/promoted.

The core idea is therefore to privilege the *Interpreted Sentence* with the maximum request "coverage". Notice that the *weighting process* changes depending whether the clause being interpreted is a condition or an action block, as the following subsections detail.

In addition the *Selection Algorithm* works under the assumption that recognition of 2 parameters over 2 has to be preferred to recognition of 2 parameters over 5, so the score derived by the number and type of the recognized parameter is normalized on the maximum score, referring to the number of total parameters.

A. Weighting an Action Block

The *Interpreted Sentences* to weight refer to the *Constructions* of the eligible *Effects* for which both verb and keywords have been recognized. So there can be more than one *Interpreted Sentence* referring to the same *Effect* that needs to be weighted.

The *Selection Algorithm* awards a determined score S_S for each strong parameters in the *Interpreted Sentences* and another score S_P, lower, for the weak and promoted parameters. The formula is

$$S = S_S * N_S + S_p * N_p$$

where N_S is the number of strong parameters in the *Interpreted Sentence* and N_P is the number of the promoted parameters. No score is assigned in case of any condition is found, because the *Interpreted Sentence* refers to a clause in an action block, so no conditions are supposed to be found. In case a conditional expression is erroneously found during the phase 3 of the interpretation process this mistake of the *Request Interpreter* has to be filtered by the *Selection Algorithm*.

The total score deriving from the parameters is normalized to 1, dividing it by the maxim score S_{MAX} that an *Interpreted Sentence* can achieve: the best case in the interpretation process is that all the parameter are found with a strong recognition, so the maximum score is given by the number of the *Construction* parameters N_{TOT} multiplied by the score assigned for a strong recognition.

$$S_{MAX} = S_S * N_{TOT}$$

This formula is valid for the *Constructions* that express a functional request (*find a cinema for a given movie*) or a direct expression of user's intentions (*I want to see this movie*), which are supposed to appear in an action block. No conditions are expected by this kind of *Constructions.* On the contrary, if a *Construction* express a statement about desired or unknown conditions concerning an object of user's interest (if this movie is shown in that cinema) the "perfect" interpretation should find not only all the input parameters but also the conditional expression expressed by the clause, so the maximum score S_{MAX} in this case is:

$$S_{MAX} = S_S * N_{TOT} + S_C$$

where S_C is the score given by the *Selection Algorithm* in case of recognition of a condition in the clause. The values of the scores (see Table I) are derived by means of experimentation and normalized taking in account the dimensions of the semantic catalogue and the dictionary.

TABLE I. RECOGNITION SCORES.

$S_S = 1$	1
$S_p = 0.5$	0.5
$S_c = 0.9$	0.9

During the analysis of the *InterpretedSentences* the *Selection Algorithm* performs a first choice between the *Interpreted Sentences* related to the same *Effect* and it chooses the one with the highest score; in case of two or more of them have the same score, the first available is chosen. It is important to notice that generally two *Interpreted Sentences* that refers to different

Constructions of the same *Effect* (requesting the same service with different words) *could* have different scores although in practice they don't compete for the selection because only the one with higher score is considered by the algorithm

Furthermore, if during this phase the *Request Interpreter* wrongly identifies a condition in the sentence, nevertheless the *Selection Algorithm* doesn't assign a score for this recognition, because the clause under analysis is in an action block. and so the *Interpreted Sentence* achieves the same not normalized score

In case of the following simple sentence (i.e., the action block of the previous sentence - see Figure 10 for details):

> *"Reserve two tickets for "Sin City" at the Romano's cinema"*

the *Request Interpreter* recognizes a *sequence* template, constituted by a single action block. The results of interpreting such sentence, are summarized in the following table (Table II).

TABLE II. INTERPRETATION RESULTS.

Effect Name	Find Cinema	Reserve Cinema Show	Find Cinema Show
Normalized Score	0,3	0,7	0,5
Total Parameters	2	4	3
Strong parameters	1	2	1
Promoted Parameters	0	1	1
Verb Recognized	False	True	False

The previous table shows how among all the movie-related services in the SSC (*FindCinema*, *FindCinemaShow* and *ReserveCinemaShow)* *ReserveCinemaShow* is the one with the highest score. In fact, despite of the common recognition of some parameters (as *cinemaName* and *filmTitle)* *ReserveCinemaShows* recognizes an additional parameter (*tickets*) thus presenting a deeper coverage of the sentence. Futhermore it is the only *InterpretedSentence* for wich a proper verbal form is found.

B. Weighting a Condition Block

The weight process of a condition block is quite similar to the one of an action block, with the difference that in this case the recognition of a conditional expression is weighed, and the correspondent score S_c is added by the Selection Algorithm to the Interpreted Sentence score, before normalizing it.

In the condition block of our example request sentence, two are the interpretation to be weighed, both referring to the Effect FindMeteoInformation, once related to the Construction 1 (which are the weather forecast for forecastDay in cityName) and the other related to the Construction 3 (if the weather forecast in cityName for forecastDay is forecast). In both cases the two input parameters are recognized, with a strong recognition of forecastDay with the value tomorrow and a promoted recognition of cityName with the value Turin. Besides, a condition on the output parameter forecast with the value rains is found during the analysis of the sentence with the Construction 3 of the Effect FindMeteoInformation, while advice contained in the Construction 1 doesn't indicate any condition to look for. The scores S_1 and S_3 assigned to the two Interpreted Sentence (before they get normalized) are:

$$S_1 = S_S * 1 + S_p * 1$$
$$S_3 = S_S * 1 + S_p * 1 + S_C$$

These have to then be normalized to the maximum scores, different for the two Interpreted Sentences, referring to different Constructions.

$$S_1 = \frac{S_S * 1 + S_p * 1}{S_S * N_{TOT}}$$
$$S_3 = \frac{S_S * 1 + S_p * 1 + S_C}{S_S * N_{TOT} + S_C}$$

with $S_3 > S_1$, assuming $C > 0$ and $S_s * N_{TOT} \geq S_S * N_S + S_p * N_p$.

This guarantees that in a condition block the Construction referring to conditional expression is preferred to the Construction referring to a direct quest.

VIII. RELATED WORK

Semantic Web Services are increasingly being experimented with to support (semi-)automated service selection and composition. For example, [5, 8, 19] describe an interactive approach to composition, in order to support users in the selection of semantically compatible operations, which can be combined in a visual workflow editor. The selection system is built upon an OWL reasoner (similar to the chaining mechanisms in our Service Generator), which interacts with a repository of service descriptors (similar to our Catalog), and on an OWL-S Process executor in charge of running workflows. [9] and [10] describe an approach to automatic composition based on a XML abstract service specification that can be assimilated to our Abstract Composition document. Those works, however, – as many others - assume that the composed VAS is specified and designed offline, and that the semantic information is used primarily to support that design work. The peculiarity of our proposition is that we introduce NLP techniques, tightly coupled with the semantic annotation and reasoning tools, in order to synthesize on demand a VAS specific to each single invocation, starting directly from an informal user request.

The closest work to the approach proposed in our paper is described in [17]: Semantic Web and NLP techniques are combined to support Web Service discovery and selection, based on information extracted from informal user input, which is processed via a network of semantic metadata [16]. However, that project focuses only on semantic service selection and dies not try to synthesize a composed VAS, complete with its control and data flow logic.

A number of other research efforts that center on semantic Web Services are complementary to our work, and delineate possible ways to strengthen it. For example, the METEOR-S Web Service Annotation Framework [14] proposes the semi-automatic semantic annotation of existing Web Services, for the purpose of discovery and composition. Currently, our descriptors are edited by hand, but we recognize the need for such an automatic annotation mechanism, which would be further important in order to annotate a VAS created on demand and enable its insertion in our SSC. GAIA [18] describes the integration of ontologies and Semantic Web technologies for service discovery and matchmaking in a pervasive computing environment. GAIA emphasizes the use of context ontologies: context is also an important part of our future research agenda, as discussed in the following section.

IX. Conclusions And Future Works

We presented an approach that allows the specification of Web Services Compositions starting from user requests expressed in controlled Natural Language. This paper shows how, under the assumptions stated in Section II, it is possible to establish a synergy between the semantic service descriptions and the Natural Language interpretation of user requests, through a common ontology and a consistent lexical vocabulary. We have presently deployed a prototypal version of the system, provided with a semantic Service Catalog in OWL-S, comprising several tenths entries, and with a limited set of logic templates able to capture a range of simple recurring logic constructs. Our preliminary experiments with the system are encouraging, since they already enable to express and synthesize significant service compositions on demand.

We are currently working to expand the Service Catalog with a wealth of information, communication and e-commerce services in order to constitute a wider source of information, thus increasing the stress and the overall noise in the recognizing and selection procedures. At the same time, we are developing a test set of user requests that focus on our SSC servicing scope, in order to establish a validation resource for further experimenting with and tuning the algorithm.

Further work is directed towards different objectives. The first one aims at reducing the overhead consequent to the introduction of new services into the SSC and consists in support to the semi-automatic creation of lexical dictionary entries associated to the SSC: by referring to external lexical resources (like Wordnet) and by formalizing the Sentence Constructions through a proper Request Ontology (see Section V for the role of *Sentence Constructions List*s within the dictionary). The second target involves the introduction in the systems of a contextual knowledge base: for example, by storing the history of serviced requests for individual users, and then by exploiting that history in order to support the interpretation process. Another major thread of research regards the broadening of the system's capability, by increasing the set of supported modular logic templates and by enabling cross reference retrieval between different clauses (*find me all the cinemas showing "Sin City" and send me the **result** by e-mail)*.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web", *Scientific American*, 2001, 284(5): 34–43.

[2] DAML.org, "OWL-S 1.1 Release", http://www.daml.org/services/owl-s/1.1/,.

[3] D. Mandell, and S. McIlraith, "Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation", *Proceedings of the Second International Semantic Web Conference*, 2003.

[4] K. Sivashanmugam, K. Verna, A. Sheth and J. Miller, Adding Semantics to Web Services Standards, *Proceedings of the International Conference on Web Services*, 2003.

[5] E. Sirin, B. Parsia, and J. Hendler, Composition-driven filtering and selection of semantic web services, *AAAI Spring Symposium on Semantic Web Services*, 2004.

[6] T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana, Business Process Execution Language for Web Services, Version 1.1, 2003.

[7] Y. Ye and G. Fisher. Supporting Reuse by Delivering Task-Relevant and Personalized Information, *Proceedings of the 24th International Conference on Software Engineering*, 2002.

[8] E. Sirin, B. Parsia, and J. Hendler, Filtering and Selecting Semantic Web Services with Interactive Composition Techniques, *IEEE Intelligent Systems*, 2004, 19(4): 42-49.

[9] B. Medjahed, A. Bouguettaya, A. Elmagarmid, Composing Web Services on the Semantic Web, *The VLDB Journal*, November 2003, 12 (4).

[10] B. Medjahed, Semantic Web Enabled Composition of Web Services, Ph. D. Dissertation, Department of Computer Science, Virginia Tech, May 2004.

[11] DAML-S, "Semantic Wes Services", http://www.daml.org/services/.

[12] WSMO, "Web Services Modeling Ontology", http://www.wsmo.org/.

[13] W3C, "WS Choreography Model Overview, 2004, http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/."

[14] A.A. Patil, S.A. Oundhakarl, A.P. Seth, and K. Verma, "Meteor-s web services annotation framework", *Proceedings of the 13th International Conference on World Wide Web*, 2004, pp. 553-562.

[15] M. Burstein, "Ontology Mapping for Dynamic Service Invocation on the Semantic Web", *Proceedings of the AAAI Spring Symposium on Semantic Web Services*, Palo Alto, 2004.

[16] MIT Media Lab, "OpenMind and ConceptNet", http://agents.media.mit.edu/projects/commonsense/.

[17] A. Faaborg, S. Chaiworawitkul, and H. Lieberman, “Using Common Sense Reasoning to Enable the Semantic Web”, http://agents.media.mit.edu/projects/semanticweb/.
[18] A. Ranganathan, R.E. McGrath, R.H. Campbell, and M.D. Mickunas, “Ontologies in a Pervasive Computing Environment”, *Eighteenth International Joint Conference On Artificial Intelligence, Workshop on Information Integration on the Web*, 2003.
[19] E. Sirin, J. Hendler, and B. Parsia, “Semi-automatic Composition of Web Services using Semantic Descriptions”, *5th International Conference On Enterprise Information Systems France - April, 2003.*
[20] B. Srivastava and J. Koehler. Web Service Composition - "Current Solutions and Open Problems". *ICAPS 2003 Workshop on Planning for Web Services*, 2003.
[21] F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in EFlow. *In Proceedings of 12th International Conference on Advanced Information Systems Engineering(CAiSE), Stockholm, Sweden, June 2000.* Springer Verlag.
[22] D. Mandell and S. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. *In Proceedings of the Second International Semantic Web Conference (ISWC2003)*, 2003
[23] Jian Yang and Mike. P. Papazoglou, Web Component: A Substrate for Web Service Reuse and Composition, Advanced Information Systems Engineering: 14th International Conference, *CAiSE 2002 Toronto, Canada, May 27-31, 2002. Proceedings*
[24] Evren Sirin, Bijan Parsia, and James Hendler. Template-based composition of semantic web services. *In AAAI Fall Symposium on Agents and the Semantic Web*, Virginia, USA, November 2005.
[25] S. McIlraith and T. C. Son. Adapting Golog for composition of Semantic Web services. *In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning(KR2002)*, Toulouse, France, April 2002.
[26] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web services. *IEEE Intelligent Systems*, 16(2):46.53, March/April 2001.
[27] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of Web service. *In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA*, May 2002. ACM.

Alessio Bosca was born in Alba, Italy in 1977. He received the M.S. degree in computer science. and the Ph.D degree in computer science from the Politecnico di Torino, Torino, Italy, in 2002 and 2006 respectively.

He is currently working as a researcher at the Politecnico di Torino and is involved in the “SPICE” EU research project, his research interests include Semantic Web (visualization of ontologies and composition of services) and Human Language Technologies.

Fulvio Corno received the M.S. degree in electronic engineering and the Ph.D. degree in computer science from the Politecnico di Torino, Torino, Italy, in 1991 and 1994, respectively.

He is currently an Associate Professor at the Politecnico di Torino. His research interests include computational intelligence, semantic web applications, assistive interfaces. He published more than 150 papers on these topics.

Giuseppe Valetto received the M.S. degree in electronic engineering from Politecnico di Torino (Turin - Italy) in 1992, and his Master of Science and Ph.D in Computer Science from Columbia University (New York, USA) in 1994 and 2004, respectively. He has worked in industrial and academic research environments since 1994.

In 2000, he joined TILAB, where he has worked in Autonomic Computing, management of mobile services, pervasive services and applications, Grid Computing and Service Oriented Architectures. He has taken part in or led several international research projects, in IST or EURESCOM.

He is currently a post-doc at IBM T.J. Watson research center in Hawthorne, NY, USA. His research interests include autonomic computing and communication, and the use of Semantic Web techniques in service-oriented platforms. He is the co-author of more than 30 scientific publications in international conferences and journals.

Roberta Maglione was born in born in Ivrea, Turin, Italy, in 1975. She received the Laurea degree in electronics engineering from the Politecnico di Torino, Turin, Italy, in May 2000. Between August 2000 and April 2004, she has been working with Cisco Systems, Inc., San Jose CA, where she focused on the design and implementation of complex telecommunication networks to deploy advanced technologies such as QoS and IP Multicast.

In May 2004, she joined Telecom Italia Research Center in Turin Italy, where she worked on a research project whose objective was the definition of the architecture and the methodologies for automated service creation, execution and delivery on SIP based Service Platforms.

She is currently working on IP Network department where she is involved in designing and testing of Broadband Network Solutions for triple-play services. Publications: M. Ajmone Marsan, G. Ginella, R. Maglione, M. Meo, "Performance Analysis of Hierarchical Cellular Networks with Generally Distributed Call Holding Times and Dwell Times", IEEE Transactions on Wireless Communications, Vol.3, No.1, pp.248-257, January 2004.