# Automatic K-Resources Discovery for Hybrid Web Connected Environments

Lara Kallab*†, Richard Chbeir‡ and Michael Mrissa§
*‡Univ Pau & Pays Adour, E2S UPPA, LIUPPA, EA3000, Anglet, 64600, France
Email: *lara.kallab@univ-pau.fr, ‡richard.chbeir@univ-pau.fr
†Nobatek/INEF4, Anglet, 64600, France
Email: lkallab@nobatek.inef4.com
§InnoRenew CoE, Livade 6, Izola, 6310, Slovenia
Email: michael.mrissa@innorenew.eu

***Abstract*—Recently, RESTful services, designed as resources, have seen their popularity rising and have shown their potential in composing reliable Web-scale environments (Web applications, Web platforms, Web of Things (WoT), etc.). However, discovering the necessary resources for a composition is becoming more challenging. This is due to 1) the growing number of published Web-based resources, and 2) the highly dynamic nature of the WoT environment, in which smart devices, connected to the Web platform, are exposed as resources. In this paper, we propose an automatic resource discovery, applicable in hybrid Web-based environments providing static linked resources always available on the Web, and connecting dynamic resources that can be connected to and removed from the Web at different time periods. The solution presents an indexing schema that makes resource discovery process fast, especially in large-scale environments. Experiments were conducted in different environment setups to test the effective performance of our approach.**

***Index Terms*—Web Connected Environments; Web of Things; Dynamic Resource; Semantic Description; Automatic Discovery**

## I. Introduction

The Web 4.0 is the newest evolution of the Web paradigm associated to the Internet of Things (IoT) concept [3], which refers to the networked interconnection of smart devices collecting and exchanging data. The Web of Things (WoT) [4] extends the IoT by integrating devices with the Web infrastructure, and exposing them as Web resources following the REST (REpresentational State Transfer) architecture style [18]. REST promotes publishing the functionalities of Web applications, Web platforms, WoT, etc., as RESTful services designed as resources. A resource can be either dynamic, i.e., connected to and removed from the Web environment at different time periods, or static, i.e., established to be always available on the Web. It provides several functions invocable using the Hypertext Transfer Protocol (HTTP) methods (e.g., GET and POST, PUT, and DELETE). However, there are cases in which a single resource is not sufficient to answer a user request, and often, combining two or more resources that form a resource composition, achieves the desired output. Although many works carried out RESTful service composition [9], discovering the necessary resources remains a challenge, and is becoming even more complex due to: 1) the highly dynamic nature of the WoT resources [6], and 2) the large number of resources connected to the Web [17]. The dynamic nature and the huge number of connected resources make their automatic discovery a necessity. Lately, automatic resource discovery has emerged as an active research area [21], however, several challenges still need to be addressed:

- **Discover dynamic resources connected at runtime**: With respect to the HATEOAS (Hypermedia as the Engine of Application State) [18] principle, hypermedia resources links are included within resources responses, during their design, to identify the next possible resources to call based on the current resource state, thus, forming a graph of linked resources. This allows automated tools to navigate through resources links to discover the upcoming appropriate resources. However, the dynamic resources connected at runtime are not linked to the existing graph resources, making their discovery a challenging task.
- **Identify K-resources**: Due to WoT dynamic aspect, dynamic resources may be unavailable for execution, even if they were identified during the discovery process. Also, some resources can be more qualified than others while answering to user requests. And sometimes, there are demands that require the coverage of several connected resources in order to be processed correctly. Thus, finding K-resources ($K \in \mathbb{N}^*$) providing the same required function is important to fulfill more efficiently user requests.
- **Make resource discovery process fast**: Numerous resources can connect to the Web, forming a large-scale Web resource environment. This makes resource discovery a complex process, especially when dealing with demands that require fast responses. Therefore, discovering resources with acceptable delays is necessary to satisfy user requests in an effective manner.

In the literature, several approaches have proposed machine-readable REST service descriptions [1][7] to allow automatic service discovery. However, these descriptions fail in considering the dynamic aspect of resources. Such aspect is also not covered in many works related to resource discovery as in [8][16][5]. Besides, most of the approaches, like [8], do not allow K-resources discovery for the same required function.
To address the aforementioned challenges, we propose in this paper a graph-based approach for automating K-Resources (KR) discovery. Our solution is generic and applicable in

hybrid Web-based environments that provide static linked resources described through a Hypermedia-based language, i.e., Hydra vocabulary [20], and connect dynamic resources. In this work, we define a formal representation that models the resources (i.e., dynamic and static) with their links into one single resource graph, and extends Hydra, which is enriched with semantic annotations (on the resources provided functions and related links) [7], to allow the description of dynamic resources. Our approach is able to find K-resources realizing the same required function, and uses an indexing schema that maps resources to their functions to make resource discovery fast, especially in large-scale Web environments. The rest of the paper is organized as follows. Section 2 presents a scenario to motivate our work, and discusses the main challenges. Section 3 gives some preliminaries related to the HATEOAS principle and Hydra vocabulary. Section 4 presents the related work, and shows the originality of our approach. Section 5 details our automatic resource discovery solution for hybrid Web environments. Section 6 evaluates the performance of the solution. Finally, Section 7 summarizes the work and gives some future directions.

## II. Motivating Scenario and Challenges

Our motivating scenario is illustrated through a Web-based management platform currently being developed under the scope of HIT2GAP[1] H2020 European project, which aims at reducing the energy gap in smart buildings [10]. More technically, the platform provides static resources for: collecting heterogeneous on-site data (e.g., internal temperature and energy consumption), preprocessing the collected data (e.g., outliers correction and data alignment), and analyzing the data (e.g., energy prediction and energy model calibration).
Each resource provides a set of functions invocable using HTTP methods. Following HATEOAS [18], HIT2GAP static resources are linked together based on their provided functions defined in a function graph (cf. Definition 1), forming a resource oriented directed graph. The links between the resources are included in each resource description, which is expressed in Hydra [20] and registered in a triplestore-based repository. To increase the quality of the collected and processed data, HIT2GAP is designed to be an open and dynamic environment that allows ad-hoc connection of external resources (e.g., mobile phones and tablets) at runtime. Such resources, mainly exposed by building occupants devices, can be connected at different time periods. The external resources, accessible through public Unified Resource Identifiers (URIs), are not registered into the repository given their dynamic aspect. Thus, HIT2GAP follows a hybrid architecture that copes with registered and non-registered resources. Several requests occur in HIT2GAP. For example, we consider the case of the building manager of one of the pilot sites, who needs to predict the heating energy consumption of a specific zone of his building (e.g., his office), for the upcoming 2 hours. To satisfy his demand, it is important to identify the resources that fulfill his prediction need. However, several challenges arise to discover the required resources, as showsn in Figure 1:

- **Discover external resources**: The dynamic nature of the HIT2GAP environment, in which external resources can be added and removed dynamically, makes it difficult for the building manager to identify the suitable resources answering his request. This is also a complex task even for the automated tools that navigate through resources links to identify the next resources to call, since dynamic resources are not linked to the existing static resources. Thus, to identify both dynamic and static resources (e.g., a resource provided by the smart-phone of the building manager to capture the ambient temperature, and an energy consumption prediction resource embedded within HIT2GAP platform), it is important to link them in a single resource model. This requires a semantic approach to allow automatic resource discovery.

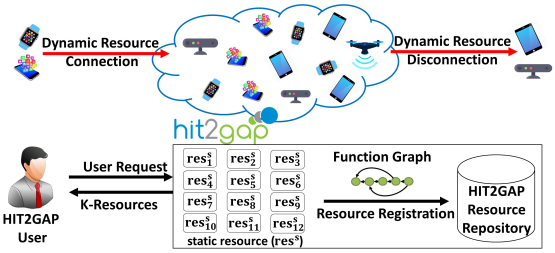

**Figure 1:** Resource discovery process in HIT2GAP

- **Identify K-resources providing the same required function**: When a dynamic resource, exposed by the building manager smart-phone for collecting the ambient temperature, is identified during resource discovery, it may be disconnected due to a restart operation occurred to the smart-phone. Thus, the provided function by the resource will not be covered. At the same time, another mobile phone can expose a resource that collects the ambient temperature more efficiently. And in some cases, the coverage of several resources at once, as collecting the temperature of a big building zone from different locations, is necessary. For these reasons, finding K-resources providing the same required function is essential.
- **Speed-up resource discovery**: Finding suitable resources in a huge Web environment with an acceptable response time, is important to answer user request efficiently. As such, the faster the energy consumption prediction of a building is, the quicker the analysis of the predicted results are, and thus, the performance of systems and devices installed within the building are well managed. Therefore, speeding-up resource discovery within large Web environments, is necessary.

To cope with these challenges, we propose a solution for the discovery of static resources supporting HATEOAS, and connected dynamic resources. This is done by 1) modeling the resources in a single resource graph, and 2) adapting graph algorithms to explore Hydra-based resource descriptions enriched by semantic annotations, and identify suitable resources

[1]Highly Innovative building control Tools Tackling the energy performance GAP: http://www.hit2gap.eu

answering to user request. Our solution discovers K-resources for the same required function, and uses an indexing schema defined to identify the most appropriate resources from which the discovery graph algorithms will start their search.

## III. Preliminaries

REST [18] has recently become a popular choice for designing Web services, due to several factors, such as client and server separation, Web services visibility, reliability and scalability [2]. REST services are resources identified by URIs that can be invoked using HTTP methods (e.g., GET, POST, PUT, and DELETE) to provide specific functions. Several principles [18] are to be considered while implementing REST services. HATEOAS, the last feature of REST, is a constraint that consists in including within resource response message, the set of resources URIs that can be called next, based on the current resource state. Although HATEOAS is still rarely used [13], our work supports it, as it allows generic clients (typically Web browsers) to dynamically navigate to the next appropriate resources. Several languages exist to describe resources while supporting HATEOAS, e.g., HAL[2] and SIREN[3]. In our work, we use Hydra [20], a lightweight vocabulary that defines fundamental concepts to describe RESTful services in a machine understandable form. Hydra is expressed via JSON-LD (JavaScript Object Notation for Linked Data) specification, which decouples the resource serialization format from the communication format between the server and the clients. JSON-LD is an easy to learn and simple format that maps resources properties, e.g., their provided functions, to concepts defined in existing data models (e.g., ontologies), allowing resources to be effectively exploited by automated tools. An example of a resource description expressed using Hydra in JSON-LD is available online[4], showing resource links annotated using semantic relation types, i.e., "isSimilar" and "isComplementary" [7].

## IV. Related Work

Our work relates to Web resource description and Web resource discovery research areas. Next, we highlight the most interesting works in these fields.

### A. Resource Description

The authors in [22] enrich resources descriptions with semantic annotations for more autonomous Web service utilization. The descriptions are expressed using OpenAPI service format(https://www.openapis.org/). Though it enables a generic and automatic client service interaction, the work covers the resources callable via "GET" method. In the future, the authors will consider the rest of HTTP verbs, and explore the integration of their method with approaches as Hydra.

The Resource Linking Language (ReLL) [1] is a model that allows providers to represent RESTful services, with emphasis on the hypermedia characteristic and linked data. Despite from being a rich data format that provides a formal definition of resources and links, it does not support the dynamic aspect of resources and link them to the existing related resources.

In [7], a resource is attached to a Hydra-based descriptor that mainly includes i) the HTTP operation used to invoke the resource, ii) the necessary inputs and the provided outputs, and iii) information about other related resources. Semantic annotations are integrated into these descriptors in [8], on the resources HTTP operations, and their links to other resources, to automate the resource discovery process. However, the descriptions represent only static resources. In our work, we extend Hydra to describe both static and dynamic resources.

### B. Resource Discovery

In [8], a resource discovery approach is proposed. It uses a BFS-based algorithm to discover resources, and explores the semantically annotated resource descriptions defined in [7] to determine if a resource suits the required functions. Apart from discovering only static resources, the discovery process finds one resource for each required function, and thus, prevents the identification of other resources providing similar functions. This is an important criteria to consider in dynamic Web environments, as it is possible to identify more qualified resources to answer user request, and substitute resources in case of non-availability. Moreover, the proposed discovery algorithm requires a given initial resource URI from the end user in order to be able to crawl the resource graph. Such task is not obvious for non-expert users. In our work, end-users express their demands simply through a single function selected from a given list. Also, our solution allows the use of different graph-based algorithms (i.e., BFS and DFS) to traverse the resource graph, and optimizes resource discovery using an indexing schema that maps resources to their provided functions.

In [16], a Web service description and interaction approach for automatic Web service discovery called RESTdesc, is proposed. The approach is based on Notation3 RDF (Resource Description Framework) syntax to describe REST services, and uses operational semantics of Notation3 in order to allow a flexible discovery. Although, it respects the HATEOAS principle, the description does not allow the discovery of dynamic resources. Moreover, the solution is used to crawl the related resources without identifying more than one resource for a required task, and depends from a complex logic language, Notation3, which is a superset of RDF. Notation3 is difficult to use, even for expert users, compared to Hydra (the language adopted in our solution) that is expressed through JSON, a comprehensible and easy to learn format for humans.

Focused on hypermedia, a service description model is proposed in [5] allowing the generation of a graph that captures state transitions in an activity layer, as well as resources, transitions, and response semantics in a semantic layer. However, the method uses a resource language that does not support dynamic resources. Moreover, the solution requires that the user knows the Schema.org data model, which has been extended with a set of concepts to semantically annotate

[2]http://stateless.co/hal_specification.html

[3]https://sookocheff.com/post/api/on-choosing-a-hypermedia-format/

[4]https://tinyurl.com/y8c6cy2o

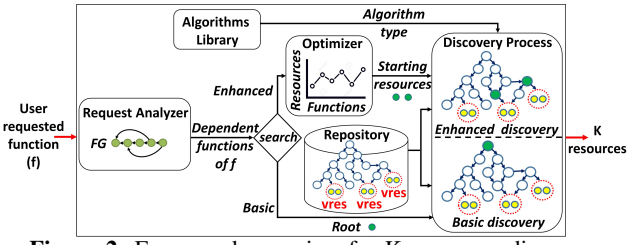


**Figure 2:** Framework overview for K-resources discovery

resources descriptions. Also, user needs are expressed through a graph query that requires knowledge from end-users.

Dynamic resource discovery is an active research area in other domains, as sensor networks [19][12], and fog computing [14][11]. However, these solutions mainly focus on the discovery of dynamic resources exposed by mobile devices that communicate with their existing neighbors, independently from their provided functions. Contrary to such approaches, our work is based on the functional aspect of the resources, which are linked together through semantic links according to their provided functions. Also, our solution combines existing linked resources (established to be always available on the Web) and connected dynamic resources into one single graph model, thus, allowing the discovery of both resource types.

## V. Automatic K-Resources Discovery Solution

In this section, we present our solution to discover automatically K-resources (static and/or dynamic) responding to user request. The latter is expressed through a single desired function, f (such as "EDP" referring to the Energy Demand Prediction function). f is selected from a generated list of functions that can be provided by the available resources connected to the Web environment at the current instant.

### A. Framework Overview

Before elaborating on our approach framework, we define a function graph, FG, containing the functions provided by the available connected resources, RES, and their dependencies (if they exist). FG is a directed acyclic graph, such that:

*Definition 1:* **FG = (F, O)**, where:

- **F** is the set of all the functions provided by RES. Formally, F=$\{f_i\}$ / $f_i \in$ res and res $\in$ RES
- **O** = $\{<\}$ is a binary order relation on F. Such binary relation is a subset of F $\times$ F linking two functions when one precedes another in the ordering. As such, if $f_1$ precedes $f_2$, it is denoted as $f_1 < f_2$. FG can also include functions that are not dependent of any other, we refer to these functions as "terminal".

Figure 2 shows the solution framework adaptive to discover K static and/or dynamic resources necessary to answer the user requested f, with f $\in$ F. When the Request Analyzer (RA) component receives f, it finds the set of functions F' (F'$\subset$ F) required to realize f. Resource search consists in finding the available connected Web resources matching F'. For this aim, we first represent all resources (static and dynamic) into one single graph, denoted as RG (cf. Definition 2), which originally contains linked static resources. To add the dynamic resources in RG, we define a virtual resource, $vres$, for each function f in F, and link it to the existing static resources realizing the same function. Each $vres$ holds the connected dynamic resources answering its correspondent function (this facilitates the integration of new functions exposed by dynamic resources in the environment). Using the RG stored in a Web-based Repository, the Discovery Process (DP) component runs a graph algorithm to identify the resources matching F'. The algorithm type is specified by the solution administrator from a library of graph-based algorithms. In this work, BFS and DFS have been implemented. Resource discovery can be 1) basic, where the algorithm starts crawling RG from the root, or 2) enhanced, where the algorithm starts RG traversal from resources pointed by a defined indexing schema presented in the Optimizer component. Such schema is built using the functions dependencies defined in FG, and the set of available linked resources described with Hydra. The indexing schema, which maps the resources to their provided functions (cf. Definition 6), returns the appropriate resources from which the discovery algorithm will begin its search, instead of traversing the resources of RG starting from the root graph.

### B. Static and Dynamic Resource-based Graph

With respect to HATEOAS, Web resources are linked together, forming a resource graph, RG. Initially, RG contains static resources established to be always available on the Web. However, the Web is a hybrid environment that allows connecting dynamic resources. Given their dynamic aspect, these resources are not linked to RG, thus, making their discovery a challenging task. To address this challenge, we define a virtual resource, $vres$, for each function f in F. A $vres$ can be permanent, $vres^p$, or temporary, $vres^t$. The $vres^p$ holds the connected dynamic resources realizing its related function that exists in F, and is linked to the static resources providing that same function via the "isSimilar" relation. As such, if a static resource provides both $f_1$ and $f_2$, it will be linked to the virtual resources, $vres^p_1$ and $vres^p_2$, defined for each of these functions respectively. And, when a dynamic resource, $res^d$, providing f is connected, it will be included in the $vres^p$ defined for f. Thus, we obtain a resource oriented directed graph, RG, combining dynamic and static resources. Generally, the functions of a connected dynamic resource are selected by the resource provider from a list containing the existing functions, F. However, if $res^d$ realizes a function that is not included in F, it will be added in a temporary virtual resource, $vres^t$, defined specifically to the new function. $vres^t$ disappears when all of its related dynamic resources are disconnected, contrary to the $vres^p$, which is always present in RG. $vres^t$ is linked to $vres^p$ through "isRelated" relation. Such linking is based on the dependencies between $vres^p$ and $vres^t$ related functions[5]. Figure 3 shows an example of the relations between the different type of resources based on

[5]Currently the new functions are added randomly in FG. Their real dependencies with other functions will be explored in subsequent work

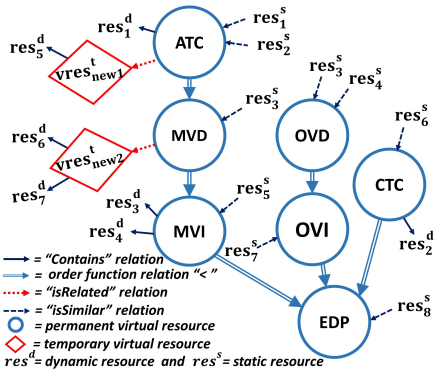


**Figure 3:** An example model of the relations between the resources

the dependencies of the functions[6] required to realize "EDP". Formally, RG is defined as:

*Definition 2:* **RG = (Root, RES, REL, f, g, t)**, where:

- **Root** is the set of any resource, $res$, that is not being pointed by any other resource in the graph. In this work, such set is formed by static resources.
- **RES** is the set of all the static, dynamic, and virtual resources connected to the Web environment.
  RES = $RES^S \cup RES^D \cup RES^V$, with:
  - $RES^S$=$\{res^s_{i\in\mathbb{N}}\}$, is the set of static resources
  - $RES^D$=$\{res^d_{i\in\mathbb{N}}\}$, is the set of dynamic resources
  - $RES^V$=$VRES^P \cup VRES^T$, is the set of permanent and temporary virtual resources, with $VRES^P$=$\{vres^p_{i\in\mathbb{N}}\}$ and $VRES^T$=$\{vres^t_{i\in\mathbb{N}}\}$
- **REL** = $R \cup C \cup T$, is the set of relations linking the resources to each other, where:
  - **R** refers to the relations used to link static resources to other resources, i.e., static or permanent virtual.
    $R = \{\simeq, <\}$, where '$\simeq$' denotes "isSimilar" relation, and '$<$' denotes "isComplementary" relation
  - **C** refers to the "contains" relation, such that C = $\{\in\}$. It is used to link virtual resources to dynamic resources
  - **T** refers to the "isRelated" relation used to link permanent virtual resources to temporary ones, such that T = $\{\rightarrow\}$
- **f** is the function linking static resources together, and static resources to permanent virtual resources, using R, such that f: $RES^S \overset{R}{\mapsto} RES^S | RES^S \overset{R}{\mapsto} VRES^P$
- **g** is the function relating virtual resources to dynamic resources, using C, with g:$RES^V \overset{C}{\mapsto} RES^D$
- **t** is the function relating permanent virtual resources to temporary ones, using T, with t:$VRES^P \overset{T}{\mapsto} VRES^T$

A resource, $res$, can be static, dynamic or virtual (permanent or temporary), and is defined as:

*Definition 3:* $\boldsymbol{res = res^s \mid res^d \mid vres^p \mid vres^t}$

Each static/dynamic resource is formally represented as:

*Definition 4:* $\boldsymbol{res^{s|d}}$ **:** $\prec$ **id, F, L** $\succ$, where:

- **id**, refers to the URI Web address used to invoke $res^{s|d}$

[6]ATC (Air Temperature Collection), MVD (Missing Values Detection), OVD (Outliers Values Detection), MVI (Missing Values Interpolation), OVI (Outliers Values Interpolation), and CTC (Climate Temperature Collection)

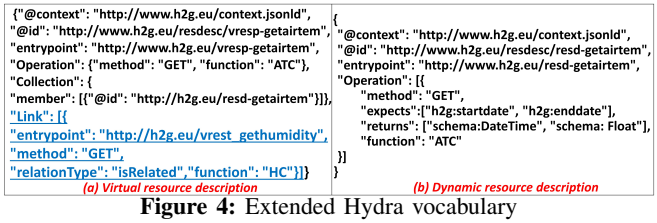


**Figure 4:** Extended Hydra vocabulary

- **F** = $\bigcup_{i=1}^{\mathbb{N}^*} \{\mathbf{f_i}\}$, designates the set of functions provided by $res^{s|d}$, such that $\mathbf{f_i}$ = $(\boldsymbol{n}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{m})$:
  - **n**, is the name of the function
  - **I** = $\bigcup_{i=1}^{\mathbb{N}^*} \{in_i\}$, is the set of the function inputs
  - **O** = $\bigcup_{i=1}^{\mathbb{N}^*} \{out_i\}$, is the set of the function outputs
  - **m** $\in$ {POST, PUT, DELETE, GET, HEAD, PATCH, CONNECT, OPTIONS, TRACE}, is the HTTP verb used to call $f_i$
- **L** refers to the set of linked resources (if they exist) to $res^{s|d}$. It is always Null for $res^d$, however, it can be defined for each $res^s$ as L = $\bigcup_{i=1}^{\mathbb{N}^*} \{l_i\}$, where:
  - $l_i$ = $(res.f, r)$, with $res.f$ is the function provided by the linked $res$, such that $res = res^s | vres^p$, and r $\in$ R

A permanent/temporary virtual resource is defined as:

*Definition 5:* $\boldsymbol{vres^{p|t}}$ **:** $\prec$ **id, f, D, L** $\succ$, where:

- **id**, refers to the URI Web address used to invoke $vres$ and access its correspondent dynamic resources
- **f** = $(\boldsymbol{n}, \boldsymbol{GET})$ refers to the function related to $vres$, with:
  - **n** is the name of the function defined for $vres$
  - **GET** refers to the HTTP verb used to call $vres$
- **D** = designates the set of dynamic resources supporting the same function defined for $vres$, with D=$\{res^d_{i\in\mathbb{N}}\}$, $res^d_{i\in\mathbb{N}} \in RES^D$, and $vres$ **C** D
- **L** refers to the set of linked resources (if they exist) to $vres^{p|t}$. It is always Null for $vres^t$, however, it can be defined for each $vres^p$ as L = $\bigcup_{i=1}^{\mathbb{N}^*} \{l_i\}$, where:
  - $l_i$ = $(vres^t.f,\ r)$, such that $vres^t.f$ is the function provided by the linked $vres^t$, and r $\in$ T

Based on the definitions, we extended Hydra to describe dynamic and virtual resources, as shown in Figure 4. Figure 4-a presents the description of a $vres$ ($vres^p$ or $vres^t$), where the "Link" field exists only for a $vres^p$. In the descriptions, the term "Operation" denotes the provided function by the resource, and the term "function" refers to a function defined in FG. The inputs and outputs of an operation are included in the "expects" and "returns" fields respectively. They are represented as key:value pair, where key is the term mapped to a specific data model, and value is the concept to which each input and output is referring to.

### C. Indexing Schema for an Enhanced Resource Search

Exploring graphs like the Web environment requires graph traversal algorithms that traverse RG nodes (i.e., resources) to find the appropriate ones realizing user request. The most popular algorithms are Depth First Search (DFS) and Breadth First Search (BFS) [15]. We conducted several tests to compare both BFS and DFS. The results[7] show that the performance

[7]Test results are available online: https://tinyurl.com/y7dq4yqk

of each algorithm depends on the functions distribution and the localization of the requested function in FG. Originally, the resource discovery algorithm starts from RG root, leading to a huge response time when dealing with large graphs. To optimize the time of the resource discovery, we define an indexing schema, IdS, as shown in Figure 5. The schema maps the resources (i.e., $RES^S$, and $VRES^T$ if they exist) to their provided functions. $RES^S$ provide the initial functions in FG, while $VRES^T$ answer newly added functions. In the schema, each function is represented by an index x $\in \mathbb{N}$ having a signature (fsignature) that includes the functions indices required to realize it. And, each resource is referenced by a value y $\in \mathbb{N}$ having a signature (rsignature) consisting of the reference values of the resources linked to it. IdS is a two dimensional space that is formally defined as:

*Definition 6:* **IdS = (o, $\mathbf{d_1}$, $\mathbf{d_2}$)**, where:

- o is the origin positioned at (0, 0). It denotes a special resource containing $RES^S$, and $VRES^T$ (when they exist). o denotes also an empty function realized by $RES^S$ and $VRES^T$.
- $d_1$=$\{x\}$ is the abscissa axis values referring to the indices of the functions F. Each x has a fsignature, such that:
  - fsignature = $\{x'\}$, with x'$\in d_1$, x'$\neq$x, and $\exists$ x'$\in$ fsignature such that x'.fsignature=$\varnothing$ denoted as "terminal".
- $d_2$=$\{y\}$ is the ordinate axis values referring to $RES^S$ and $VRES^T$. Each y has a rsignature, such that:
  - rsignature=$\{y'\}$, with y'$\in d_2$ and y'$\neq$y

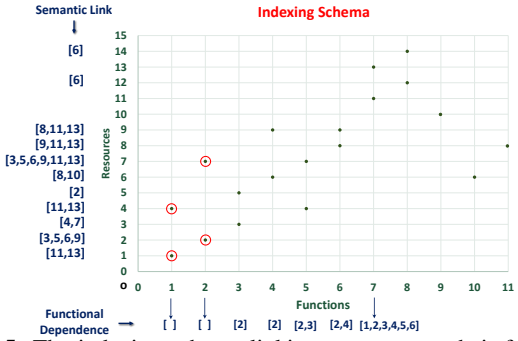


**Figure 5:** The indexing schema linking resources to their functions

In Figure 5, $f_7$ is preceded by 6 functions, [1,2,3,4,5,6]. Based on the analysis of these functions signatures, $f_1$ and $f_2$ are terminals. The resources realizing such two functions are $r_1$, $r_2$, $r_4$, and $r_7$ (circled in red), and will act as initial resources from which the discovery algorithm will begin its process, instead of starting from the graph root. The construction of IdS requires 1) the definition of FG to get the functions with their correspondent signatures, and 2) the existence of linked $RES^S$ and $VRES^T$ (if they exist) described with Hydra to get their provided functions with their related resources (if defined). The performance evaluation of IdS construction[8] show that the response time and memory usage increase with the evolution of both functions and resources numbers. Such evolution is huge when the number of functions and resources is high. Thus, updating the IdS dynamically without regenerating it again is an improvement that we seek to do in the future.

[8]The evaluation results are available online: https://tinyurl.com/ydbaubtp

### D. Automatic K-Resources Discovery Process

Our discovery solution identifies static and/or dynamic resources, and adapts several graph-based algorithms for RG traversal. The algorithm to be used is given as input to the resource discovery process based on a library of algorithms (i.e., BFS or DFS in this work), which will be extended in future works. The discovery process uses an integer variable, K, defined by the solution administrator, to represent the maximum number of discovered resources realizing similar functions. Algorithm 1 presents the pseudo code of the defined discovery process having the following entry data:

- **algoType** (string): denotes the algorithm type to be used
- **F'** (array of string): F'$\subset$ F, contains the requested function f and the functions that precedes f in FG
- **Id** (array of string): refers to the resource(s) id(s) from which the specified algorithm type starts its process
- **K** (integer): refers to the maximum number of discovered resources providing similar functions

The output is the **discovered** array, containing the pairs [**f**, **id**] that correspond to the discovered resources matching each function f $\in$ **F'**. Based on the given **algoType**, the discovery process runs the corresponding algorithm (**runAlgoType()**) that explores RG starting from the resources included in the **Id** array. **currentId** refers to the resource that is being processed, which is initially the first resource of the **Id** array. For each unvisited resource, the algorithm gets the relative Hydra description through **getDesc()** (line 7). If the operation function matches one of the functions in **F'** using the **functionMatch()** (line 9), the algorithm checks whether the resource is virtual or static (lines 10 to 16). When it is virtual, the ids of the dynamic resources included in the description (line 12) are inserted with their relative function in the **discovered** array. When it is static, the corresponding resource id is stored in the **discovered** array with the relative function (line 16). If the number of the discovered resources realizing the current function is equal to **K** (lines 13 and 17), the function is removed from **F'**. Such number is calculated using the **resFound()** that we implemented apart. To explore other resources, the algorithm follows the resource semantic annotated links (lines 19 to 21). The **resToExplore** and **visited** arrays refer, respectively, to the set of resources ids that will be explored next, and to the ids of the resources already crossed. The algorithm keeps running until **F'** is empty, denoting that K-resources realizing each required function in **F'** are discovered.

## VI. Evaluation and Discussion

In this section, we evaluate the performance of our solution in different function and resource graphs topologies, by varying, for example, the number of functions and the number of resources providing, each, one function included in FG. In the tests, we focus on studying our approach in 2 different forms: 1) basic, where the search starts from the graph root of RG, and 2) enhanced, where the search starts from the resource(s)

**Algorithm 1:** Pseudo code of the automatic KR discovery process

```
1  visited, currentId : array of string
2  currentId = id[0]
3  runAlgoType(): // the execution of the algorithm corresponding to the given algoType
4  while not F'.empty() do
5      if not currentId in visited then
6          visited.insert(currentId)
7          Descriptor desc = getDesc(currentId) foreach operation in desc.Operation do
8              foreach f in F' do
9                  if functionMatch(operation.function, f) then
10                     if not desc.RESD.empty() then
11                         foreach adhoc in desc.RESD do
12                             discovered.insert([f, id])
13                             if resFound(discovered, f) = K then
14                                 F'.remove(f)
15                     else
16                         discovered.insert([f, currentId])
17                         if resFound(discovered, f) = K then
18                             F'.remove(f)
19         foreach link in desc.Link do
20             if (link.relationType = isSimilar or link.relationType = isComplementary or
               link.relationType = isRelated) then
21                 resToExplore.insert(link.entrypoint) // stores the resources linked to the current
                     traversed resource
22         currentId = resToExplore.select() // selects the next resource to explore
23     else
24         currentId = id.next()
25 return discovered;
```

pointed by IdS, without considering the best algorithm type to use at each function graph topology (this will be done in another work). Therefore, we only show the experiments using one algorithm type, i.e, the BFS. To consider worst case scenarios, the tests consist of dynamic resources providing functions existing in FG, regardless of whether they were originally defined or newly added.

### A. Environment Setups

The function and resource graphs are based on simulations that are dynamically generated, using several criterion, i.e., number/order of functions, number/type of resources, and number of resources (K) providing the same function. This is done to study our approach in different functions/resources graphs topologies. In the tests[9], conducted on a Linux Debian (64 bits) virtual machine, with 1 dedicated Intel® Core™ i7-46000 CPU @ 2.10GHz 2.70GHz processor and 1 GB RAM, we show the algorithm response time (in milliseconds) based on an average of 5 sequential executions.

### B. Evaluation

Due to the lack of standard benchmark to evaluate the related work [8][16][5], we propose in this paper, the following aspects for our resource discovery approach evaluation:

- **Dynamicity**: the ability to identify appropriate resources for a user request in a dynamic environment connecting both dynamic and static resources
- **Multiplicity**: the ability to discover K-resources responding to the same required function
- **Efficiency**: the ability to identify suitable resources in a large Web environment with acceptable response time
- **Scalability**: the ability to identify resources in large graphs containing resources that provide numerous different functions

[9]The prototype code is available online: http://tinyurl.com/y7e78n24

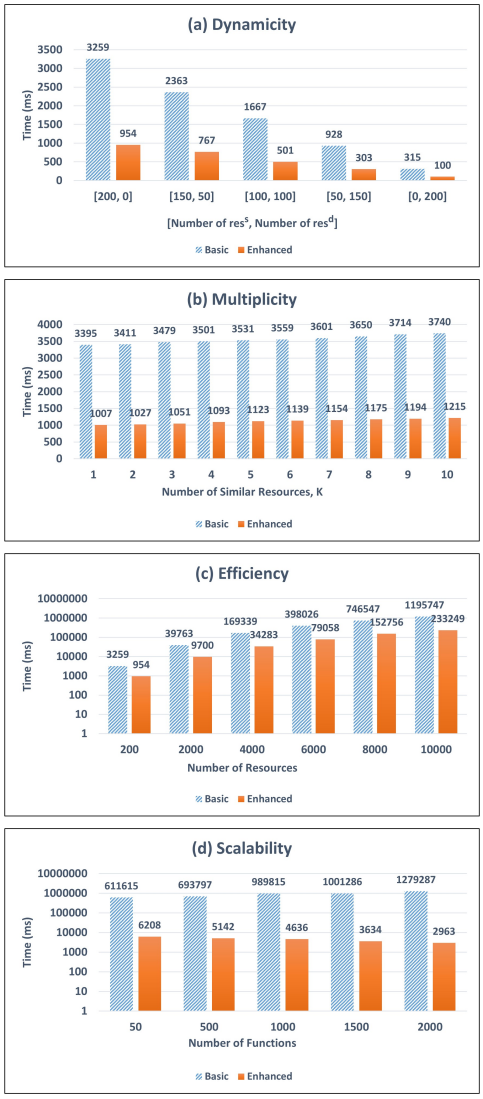


**Figure 6:** Performance results

For each aspect, we generated results by running the basic and the enhanced search forms of the BFS algorithm. For the **dynamic aspect**, the tests were executed on 5 resource graphs containing, each, 200 resources, based on a FG with 20 ordered functions. The number of dynamic and static resources varies from a graph to another with the variation of K. As such, the graph consisting of 200 static resources ([200,0]), contains 10 resources (K=10) for each function in FG. However, graph [100,100] includes 5 static and dynamic resources (K=5 for each type) realizing the same function. The aim of the tests was to find 1-resource for each function required to answer "EDP". Figure 6-(a) shows that the presence of dynamic resources reduces the response time of the resource discovery in both algorithm forms. This is explained by the existence of virtual resources from which the algorithm can access several dynamic resources realizing the same function at once. For the **multiplicity aspect**, we built a graph of 400 resources (200 static and 200 dynamic), and run our tests while varying K. Figure 6-(b) shows that the response time increases with the evolution of K in both algorithm forms. This is due to

the additional resources that the discovery process will have to find realizing a single required function. However, the results are more satisfactory using the enhanced search. For the **efficiency aspect**, Figure 6-(c) shows that the response time of the enhanced search is better than the basic one, when increasing the number of resources from 200 to 10000[10]. This highlights the utility of the indexing schema in large Web graphs. As for the **scalability aspect**, we fixed the number of resources to 10000, and varied the number of functions (from 50 to 2000). In the first 3 tests, RG graphs consist of K dynamic and static resource for each function, with K decreasing from 100, 10, and 5 respectively. In the rest 2 tests, K is defined unequally between static and dynamic resources of the RG graphs. As such, when the number of functions is 2000, RG graph includes 2 dynamic resources (K=2) and 3 static resources (k=3) for each function. Figure 6-(d) shows an increase in the response time with the evolution of the functions number with the basic search. This is due to the variety of resources providing numerous functions that are different from the required ones. However, the response time decreases in the enhanced search. This is explained by the reduction of the number of resources providing the necessary functions related to user request, since the functions number increases while the total resources number is fixed.

Figure 6-(a) results highlight the benefit of defining virtual resources containing dynamic resources realizing the same function. It facilitates and speeds-up the access to K-resources at once during resource discovery. The tests in Figures 6-(b), 6-(c) and 6-(d) show that the time curve in both algorithm forms generally increases with the evolution of the functions number, the resources number, and the similar resources number, K. Except for the scalability aspect, and with the enhanced search, the time curve decreases. This is due to the fixed number of resources while increasing the number of the provided functions, which leads to a decrease in the number of resources providing the required functions necessary to realize user request. Our experiments prove that using our indexing schema optimizes resource discovery in all graphs setups. This can be seen through the difference between the results of the basic and the enhanced searches, especially when the resources number is high (10000 resources) as shown in Figures 6-(c) and 6-(d).

## VII. Conclusion

This paper presents an automatic resource discovery approach for hybrid Web environments providing linked static resources and connecting dynamic resources. The solution uses an original indexing schema that enhances resource search in large environments, and discovers K-resources realizing the same required function. Experiments were conducted to evaluate our solution on 4 aspects: Dynamicity, multiplicity, efficiency, and scalability. In the future, we plan to study the solution performance in more complex setups (e.g., varying simultaneously the number of resources and provided functions), and test it in a real environment. Also, we aim to integrate other graph algorithms, and propose dynamically the most suitable one according to the current function graph topology. Moreover, we seek to study the measures that define the dependencies between the new and the existing functions, and update the indexing schema dynamically without regenerating it from scratch every time.

[10]Each graph contains the same exact number of static/dynamic resources

## Acknowledgment

HIT2GAP project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 680708. The authors acknowledge that the development work is carried out in a complementary manner with SIBEX: a French project funded by the Energy Transition Institute INEF 4.

## References

[1] Rosa Alarcon and Erik Wilde. From restful services to rdf: connecting the web and the semantic web. *arXiv preprint arXiv:1006.2718*, 2010.

[2] Subbu Allamaraju. *Restful web services cookbook: solutions for improving scalability and simplicity*. " O'Reilly Media, Inc.", 2010.

[3] Fernando Luis Almeida. Concept and dimensions of web 4.0. *International Journal of Computers & Technology*, 16(7):7040–7046, 2017.

[4] Payam Barnaghi, Amit Sheth, and Cory Henson. From data to actionable knowledge: Big data challenges in the web of things [guest editors' introduction]. *IEEE Intelligent Systems*, 28(6):6–11, 2013.

[5] Alarcon Rosa et al. Rest web service description for graph-based service discovery. In *ICWE*, pages 461–478. Springer, 2015.

[6] Aziez Meriem et al. Service discovery for the internet of things: Comparison study of the approaches. In *Control, Decision and Information Technologies (CoDIT), 2017*, pages 0599–0604. IEEE, 2017.

[7] Bennara Mahdi et al. An approach for composing restful linked services on the web. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 977–982. ACM, 2014.

[8] Bennara Mahdi et al. Semantic-enabled and hypermedia-driven linked service discovery. In *MEDI*, pages 108–117. Springer, 2016.

[9] Garriga Martin et al. Restful service composition at a glance: A survey. *Journal of Network and Computer Applications*, 60:32–53, 2016.

[10] Lara Kallab et al. Hit2gap: Towards a better building energy management. *Energy Procedia*, 122:895 – 900, 2017. {CISBAT} 2017.

[11] Liu Wei et al. Adaptive resource discovery in mobile cloud computing. *Computer Communications*, 50:119–129, 2014.

[12] Marin-Perianu Raluca et al. Prototyping service discovery and usage in wireless sensor networks. In *32nd IEEE Conference on Local Computer Networks (LCN 2007)*, pages 841–850. IEEE, 2007.

[13] Neumann Andy et al. An analysis of public rest web service apis. *IEEE Transactions on Services Computing*, 2018.

[14] Rejiba Zeineb et al. F2c-aware: Enabling discovery in wi-fi-powered fog-to-cloud (f2c) systems. In *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 113–116. IEEE, 2018.

[15] Russell Stuart Jonathan et al. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[16] Verborgh Ruben et al. Description and interaction of restful services for automatic discovery and execution. In *AFMS 2011*, 2011.

[17] Wang Jian et al. A web service discovery approach based on common topic groups extraction. *IEEE Access*, 5:10193–10208, 2017.

[18] Roy T Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[19] Simon Jirka, Arne Bröring, and Christoph Stasch. Discovery mechanisms for the sensor web. *Sensors*, 9(4):2661–2681, 2009.

[20] Markus Lanthaler and Christian Gütl. Hydra: A vocabulary for hypermedia-driven web apis. *LDOW*, 996, 2013.

[21] Debajyoti Mukhopadhyay and Archana Chougule. A survey on web service discovery approaches. In *ICCSEA*, pages 1001–1012. Springer, 2012.

[22] Cong Peng and Guohua Bai. Using tag based semantic annotation to empower client and rest service interaction. In *COMPLEXIS 2018*, pages 64–71, 2018.