Retrieving Sensors data in Smart Buildings through Services: a similarity algorithm

Claudia Foglieni, Mirjana Mazuran, Giovanni Meroni, and Pierluigi Plebani

Politecnico di Milano – Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza Leonardo da Vinci 32, 20133 Milano (Italy)
foglieni.claudia@gmail.com, mirjana.mazuran@polimi.it,
giovanni2.meroni@mail.polimi.it, pierluigi.plebani@polimi.it

Abstract. This paper proposes a semantic-based retrieval algorithm that allows the pervasive service system to find services able to return data about specific physical phenomenon (e.g. temperature, humidity), in a given location, with particular timeliness. This retrieval algorithm can be used to increase the capabilities of a self-managing pervasive systems, with specific focus on smart buildings, by providing a flexible solution to find sensors similar to a one that failed, or to find sensor data able to control actuators.

1 Introduction

The diffusion of wireless and wired sensor networks led to the development of pervasive systems. An example are *smart buildings*, where sensors are deployed in residential, commercial and industrial buildings to monitor and control the installed devices. Service oriented architectures have been adopted to implement these pervasive systems and to hide the technical heterogeneity and the complexity of the deployed devices [1]. In this way, sensors are seen as services and the communication with them can be performed as a service invocation. Especially when heterogeneous sensor networks are considered, this pervasive service system can overcame the limitations caused by the vendor lock-in problem.

Goal of this paper is to propose an algorithm for retrieving sensors, seen as services, able to return data about a specific phenomenon, in a given location, with particular timeliness. This algorithm is based on (i) a semantic-based service description model that extends OWL-S [1] with SensorML [2] and information about the sensor location, and (ii) a similarity function that compares the output of the services with the characteristics of the needed data. Generally speaking, with this retrieval algorithm we aim to increase the capabilities of a building-related pervasive service system with the possibility to find sensors similar to another one by comparing the services that represent such sensors. Indeed, in case a sensor recording the temperature fails or is not present, with the proposed algorithm it is possible to find the closer temperature sensor and to use the data recorded by it, thus overcoming the lack of the desired information.

The rest of the paper is organized as follows. Section 2 introduces an overview of the approach. Section 3 focuses on the retrieval algorithms that represents

[1] http://www.w3.org/Submission/OWL-S/
[2] http://www.opengeospatial.org/standards/sensorml

the core of this paper. The validation of the algorithm is discussed in Section 4. Finally, after a discussion of related work in Section 5, Section 6 concludes the work and outlines some possible future work.

2 The overall approach

In this work we assume that a pervasive system is composed of many heterogeneous sensors and actuators that are spatially distributed. These devices belong to different technologies, have different semantics, and are characterized by different complexity levels. Therefore, in order to manage the gathering and processing of data we need a middleware that will hide the complexity of this scenario while allowing users to exploit all its potential. To reach this goal we use PerLa [2] to manage the data through a database abstraction, that is, the data gathered by sensors are seen as if arranged into a database and the final user is provided with a user-friendly language, featuring an SQL-like syntax, to handle it. On top of this middleware, a "Sensors as a Service" layer exposes only the data that are defined as externally accessible, thus the user or application that invokes the services relies only on the methods specified in the service interfaces without knowing the data model or the storage technology.

In this scenario, a proper service description is crucial, as it is the only way to know what a service offers and how to interact with it and, in this work, such a service description is based on a semantic characterization. In particular, we describe each available service in terms of the information that can be useful to the users to express the requirements that have to be satisfied during the retrieval process. We consider fundamental the following three dimensions: (i) *sensor*, services are related to sensors or actuators and have different characteristics (e.g., type, measure unit, etc.); (ii) *location*, sensors providing services are bound to a physical location. By knowing this location we can answer queries in a context-aware fashion; (iii) *device*, it is our aim to monitor consumption flows, thus, we need to have some information about the devices (considering the sensors themselves) that consume or produce energy.

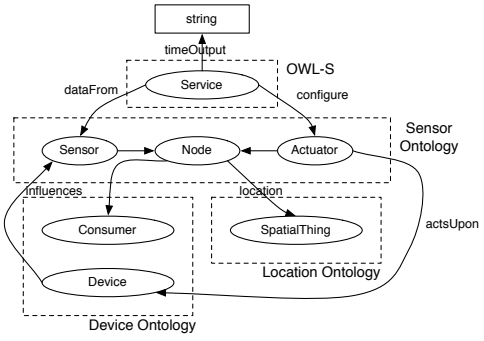

Fig. 1. Service Ontology extending OWL-S specification with sensor-specific concepts.

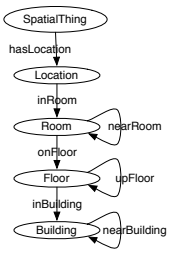

Fig. 2. Location Ontology made of hierarchies of elements in space.

To support these requirements, we adopt a service ontology. We start from the well-known OWL-S ontology and extend it with information related to our needs and application domain. Therefore, as shown in Figure 1, we enrich OWL-S with three more ontologies, each one describing one of the three dimensions introduced above. In particular, notice that each service is described in terms of:

- its temporal output (i.e. *timeOutput* relation): data returned by a service may be: i) the current data reading (*Last One*), ii) all the readings in a specific interval of time (*Interval*) or iii) the last X readings (*Last X*, where X depends on the service);
- the features of the node in the sensor network to which the service is associated (described by the *Sensor* ontology)
- the physical location of the sensor providing the service (described by the *Location* ontology as shown in Figure 2): a hierarchy ($Location \rightarrow Room \rightarrow Floor \rightarrow Building$) is used to support different granularities and different relations (*nearRoom*, *upFloor* and *nearBuilding*) are used to understand if the concepts of same granularity are somehow near one to the other;
- the features of the devices that influence or are influenced by the node to which the service is associated (described by the *Device* ontology): we specify the energy they consume or produce.

This semantic representation allows us to describe the structure of services and to reason on the data we have, in order to infer some new knowledge that might be useful during the query answering process. To better understand, let consider the following example: suppose that a user wants to know the temperature in Room 23 but that one does not have a sensor installed, yet Room 24, the one next to Room 23, has one. By having this information expressed in the location ontology, our retrieval algorithm will be able to suggest to the user the service returning the temperature in Room 23, even though a penalty will be applied to this one since it does exactly satisfy the user request.

3 Retrieval algorithm

There are many approaches supporting the service retrieval process (as discussed in Section 5). For our purposes, we assume that the users' needs are expressed in terms of a desired service, i.e., using the same description model used to describe a service the user can define the features of a service that the user is looking for. Then the description of the desired service is compared with all the available services. Formally, we introduce the following definitions:

- $\Sigma = \{\sigma_i\}$ is the set of available services, where each σ_i is described in terms of the service ontology discussed before. Moreover, $\sigma_i.values$ is the set of values that have been sensed by the sensor associated to the service σ_i.
- $\overline{\sigma} = \langle type, location, time, output \rangle$ is the desired service, defined in terms of one or more concepts and properties of the service ontology. More precisely:

- *type* is associated to the nature of the sensor, i.e., what the sensor measures (e.g., temperature, power, and so on).
- *location* defines where the data have been sensed using one of the *SpatialThing* related concept in the location ontology.
- $time = \{LastOne, LastX, Interval\}$ specifies the nature of the sensed data returned as output of the service. In case of $time = LastOne$, this means that the user is interested in the more recent sensed value, whereas $time = LastX$ in a set of more recent values. Finally, in case of $time = Interval$ the user specifies the time range in which the sensed values are considered relevant.
- Depending on the value of *time*, *output* specifies the value of $X \in \mathbf{N}$, or the initial and final hours and dates, and the granularity.

– $\overline{\Sigma} \subseteq \Sigma$ is the set of services relevant with respect to the desired service.

With respect to the traditional approaches our algorithm considers the location as a first class citizen and the similarity algorithm strongly depends on it. Moreover, having the *time* and *output*, allows the user to insert in query elements that makes more expressive the request in the reference scenario.

In our approach, we provide a similarity function $sim(\overline{\sigma}, \sigma_i) \rightarrow [0, 1]$ that compares two service descriptions and the higher the returned value the more similar the two services.

Listing 1 Similarity algorithm overview

```
Σ̄ = ∅
for all σ_i ∈ Σ do
  if σ_i.type = σ̄.type then
    sim = 1.0
    if σ̄.location ≠ ∅ then
      sim = sim * (1-location_penalty(σ_i.location, σ̄.location))
    end if
    if σ̄.time ≠ ∅ then
      sim = sim * (1-time_penalty(σ_i.time, σ̄.time))
      sim = sim * (1-output_penalty(σ_i.time, σ_i.output, σ̄.time, σ̄.output))
    end if
    if sim > th then
      Σ̄ = Σ̄ ∪ σ_i
    end if
  end if
end for
return Σ̄
```

The resulting algorithm is presented, in its main steps, in Listing 1. Here the functions *location_penalty*, *time_penalty*, and *ouput_penalty* are in charge of computing the distance between the request and the offer with respect to a specific aspect. The similarity is computed as the product of all the penalties. In this way, a good value of similarity can be obtained only if all the requirements

are matched at least partially. Indeed, it is enough that at least one of the requirements is not properly supported to significantly reduce the similarity. Finally, as also reported in the algorithm, as the elements composing $\overline{\sigma}$ are not mandatory, the corresponding penalty function could not be invoked.

3.1 Location penalty

The computation of the location penalty identifies the distance between the concepts $\overline{\sigma}.location$ and $\sigma_i.location$ in the location ontology and it is based on the following assumptions:

- If $\overline{\sigma}.location = \sigma_i.location$ no penalty is applied.
- The more distant $\overline{\sigma}.location$ to $\sigma_i.location$, the grater the penalty.
- The wider the location $\overline{\sigma}.location$ with respect to the location $\sigma_i.location$, the greater the penalty. For instance, if the user asks for a temperature in a room and the service monitors the entire floor, then the penalty will be lower than that of a service returning the temperature of the entire building.
- If the location $\overline{\sigma}.location$ is more narrow than the location $\sigma_i.location$ no penalty is applied.

Listing 2 Location algorithm

```
award=1
for all edge: edges in shortest path between σ_i.location and σ̄.location do
  if edge.from is more specific σ̄ then
    award = award * λ
  end if
end for
location_penalty = 1 - award
```

Based on these assumptions, Listing 2 shows the complete algorithm for computing the location penalty. First, the shortest path between the concepts $\overline{\sigma}.location$ and $\sigma_i.location$ is computed. As the penalty depends on the edges from less specific concepts to less specific concepts, we firstly count the opposite to compute the award. Then, the penalty is given by the opposite. The parameter λ quantifies the amount of award (penalty) to be given for each relevant hop and its value is defined after a tuning phase, that is presented in the next section.

3.2 Time penalty

Time penalty depends on the degree of compatibility between services having different time output property values. For example, if $\overline{\sigma}.time = LastX$ a service σ_i is fully compatible if $\sigma_i.time = LastX$ but, we also consider a compatibility (with a penalty) also in case $\sigma_i.time = Interval$. Indeed, the time range can cover the required more recent values. On the contrary, if $\sigma_i.time = LastOne$ cannot be considered compatible as it returns only the most recent value over the required X values. Accordingly, we defined three levels of compatibility, i.e.,

Mismatch ($penalty = 1.0$), Close ($penalty = \tau$), Exact ($penalty = 0.0$) and Table 1 reports how these penalties are associated to each possible combination of $\sigma_i.time$ and $\overline{\sigma}.time$, where the value of τ is set during the tuning phase.

		$\sigma_i.time$		
		Last One	**Last X**	**Interval**
$\overline{\sigma}.time$	**Last One**	Exact	Close	Close
	Last X	Mismatch	Exact	Close
	Interval	Mismatch	Close	Exact

Table 1. Time compatibility matrix

3.3 Output penalty

The last computed penalty is related to timeliness, that is defined as the extent to which data are timely for their use or as the property of information to arrive early or at the right time [3].

In our case, timeliness depends on the distance between when the sensed data is available with respect to when it is required, so the penalty depends on how much the timeliness is not satisfied. As the user with $\overline{\sigma}.time$ when the required data is relevant in three different ways (i.e., *LastOne*, *LastX* , and *Interval*) three different approaches for computing penalties are proposed. Differently from the previous penalties, the computation of the output penalty requires the interaction with the services. Indeed, the information available in the service description is not enough as we need to have information on the data returned by the services which can be obtained only bu invoking them. As a consequence, in this phase the services need to be invoked and the returned data analyzed. We remind that $\sigma_i.values$ represents these values and each of them is a pair of a timestamp (when the values is sensed) and the sensed value.

Listing 3 Last One algorithm

for all $\sigma_k \in \Sigma$ **do**
  max($\sigma_k.values.timestamp$) = date of the more recent sampled data
  if $max(\sigma_k.values.timestamp) <$ less_recent **then**
    less_recent = $max(\sigma_k.values.timestamp)$
  end if
  if $max(\sigma_k.values.timestamp) >$ more_recent **then**
    more_recent = $max(\sigma_k.values.timestamp)$
  end if
end for
pen_current = $1 - (max(\sigma_i.values.timestamp) - less\_recent)/(current\_date - less\_recent))$
pen_morerec = $1 - (max(\sigma_i.values.timestamp) - less\_recent)/(more\_recent - less\_recent))$
output_penalty = w_{curr} * pen_current + $w_{morerec}$ * pen_morerec

LastOne If $\overline{\sigma}.time = LastOne$ then the user is interested in services that return the most recent values. Here the discussion is on the meaning of "recent value": is it with respect to when the user submits the query, or is it the most recent among the data returned by the services? As both the solutions are valid, we consider both the situation and, using a weighted sum, we leave to the user the possibility to specify which is the best interpretation. This results in the algorithm reported in Listing 3. First of all, we assume that the *current_date* is given (for instance, by invoking a system call); then, the *more_recent* and *less_recent* dates are obtained by calling the services available and considering the lower and higher values for the returned timestamps. Then, the penalty in both the cases is calculated as a proportion of the distance between the date of the returned value and the reference date.

Last X In case of $\overline{\sigma}.time = LastX$ the user also specifies in the query the desired number (e.g., $\overline{\sigma}.output.X$) of output values. On this basis, two main aspects will be considered in the computation of the penalty:

- The number of returned values (*pen_number*): a σ_i able to retrieve at least $\overline{\sigma}.output.X$ values has lower penalty than a σ_i that satisfies the user request only partially returning a number of values lower than $\overline{\sigma}.output.X$: i.e., $pen\_number = \text{count}(\sigma_k.values) \ / \ \overline{\sigma}.output$.
- The timeliness of the returned values (*pen_recent*): a σ_i returning values that are more recent has lower penalty than a σ_i that returns values with lower timestamps: i.e., $pen\_recent = \text{lastOne}(\sigma_i)$

Having these values: $output\_penalty = 0.5 \cdot pen\_number + 0.5 \cdot pen\_recent$

Interval The third possible kind of query on the output is $\overline{\sigma}.time = Interval$, i.e, the user wants services that have sampled data in a specified time range ($\overline{\sigma}.output.start\_date$, $\overline{\sigma}.output.end\_date$). Here, the more covered is the interval, the more similar the service. To properly consider also a homogeneous coverage of the interval, a third input parameter named $\overline{\sigma}.output.granularity$ is required that specifies the sampling time inside the time range. For instance, having a time range of $10mins$ with a granularity of $60secs$, means that the ideal service should have at least 10 sampled data distributed every $1min$. Listing 4 reports the details of the adopted algorithm. First of all, given the time range, the number of subintervals is computed. Then, for each of them, we verify how many intervals are covered by the $\sigma_i.values$. Finally, the higher the number of not covered intervals, the higher the penalty.

4 Validation

To validate the proposed algorithm, we based our tests on a testbed containing data collected from 54 sensors deployed in the Intel Berkeley Research lab [3]. The sensors collected timestamped values of humidity, temperature, light and voltage, producing 2.3 million readings in a period of a month and a half.

[3] `http://db.csail.mit.edu/labdata/labdata.html`

Listing 4 Interval algorithm

```
interval = σ̄.output.end_date − σ̄.output.start_date
subintervals= interval /σ̄.output.granularity
covered_subintervals = 0
for all subintervals in interval do
  if count(σ_i.values ∈ subinterval) > 0 then
    covered_subinterval ++
  end if
end for
output_penalty = 1- (covered_subinterval / subintervals)
```

Query	$\overline{\sigma}.location$	$\overline{\sigma}.time$	$\overline{\sigma}.output$
$\overline{\sigma}_1^{tuning}$	-	Interval	Range: from 28/2/2004 6:30 AM to 28/2/2004 2:30 PM Granularity: 1 hour
$\overline{\sigma}_2^{tuning}$	Floor22	LastX	Samples: 14
$\overline{\sigma}_3^{tuning}$	Room27	LastOne	-
$\overline{\sigma}_1^{test}$	-	Interval	Range: from 28/2/2004 10:30 AM to 28/2/2004 6:30 PM Granularity: 1 hour
$\overline{\sigma}_2^{test}$	Floor12	LastX	Samples: 11
$\overline{\sigma}_3^{test}$	Room111	LastOne	-

Table 2. Tuning and test queries.

Before running the test cases to validate the approach, a proper tuning of the parameters th, λ, and τ is required. To this aim, the first three queries reported in Table 2 have submitted to the testbed varying, for each run, the values of these three parameters. For each of these queries, the list of relevant services is manually created to be used for computing the precision and recall.

Based on the obtained results, the better precision and recall values for these queries is given for the following values: $th = 0.5$; $\lambda = 0.9$; $\tau = 0.8$. The corresponding precision recall chart for this tuning set is shown in Figure 3. With this configuration, a second set of queries (some of them are reported in the lower part of Table 2) has been used to calculate the final precision and recall graph. As shown in Figure 4, the proposed algorithm properly responds to the queries with a good precision for all the percentage of recall. Indeed, when all the relevant services are returned (i.e., recall equals to 100%) the precision remains greater than 60%. To the best of our knowledge, there are no comparable approaches, thus, a comparison with existing methods is not possible at this stage.

5 Related work

Different similarity algorithms have been proposed that allow to compare sensors. In [4] a methodology to discover service similarity is based on testing and requires the evaluation of the outputs produced by the services after invoking them. With respect to this work, we are interested in introducing a semantic dimension into the matchmaking process. Research has put much effort in de-

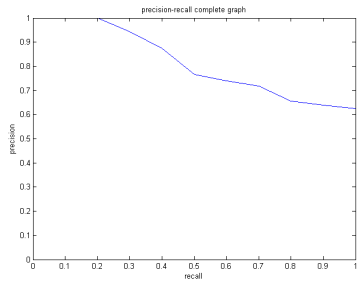

Fig. 3. Performances with the tuning set queries.

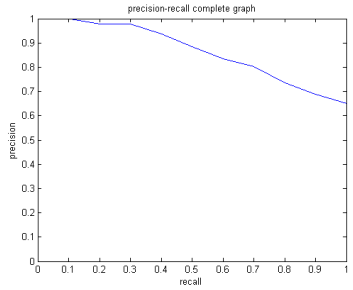

Fig. 4. Performances with the test set queries.

veloping ontology-based models to support pervasive systems [5] which mostly adopt ad hoc solutions that depend on the application domain but also bring up the need for ontologies to describe sensors [6] and locations [5] in particular. Among sensor ontologies it is worth pointing out the Semantic Sensor Web, a framework to support semantic annotation of sensor data through the use of ontologies to elicit the features of the sensors. In this work we make use of such an ontology to extend the OWL-S ontology with useful domain-dependent information similarly to what has been done in the context of the SemsorGrid4Env EU project [4].

Other works [7, 8] enrich services description by associating their inputs and outputs with the concepts in a domain ontology and adopt a hybrid strategy that exploits both logic-based reasoning and content-based information retrieval techniques. Authors in [9] adopt an ontology to describe the interactions between the processes in a service and then compute the similarity by keeping into account: i) the structural similarity, based on the outputs of the services and ii) the semantic similarity, based on the data in the ontology.

As for structural similarity, in [10] and [11] two approaches are described that dynamically select and recommend services to users. However, both approaches are based on historical data and thus they assume that the similarity computed in the past can be used to refine the future rankings.

On the other hand, different approaches have been proposed that exploit an ontology to determine semantic similarity between services. The most common approaches are distance metrics, information-based measures and more complex ontology frameworks. In [12] authors propose a information-based similarity measure that also takes into account reasoning, that is, the semantic similarity of an object depends on how many new objects it can generate. In [13] the authors take a similar road but propose a weighted algorithm that takes into account the frequency of words appearing the in semantic description of services. In this work we focus on a simpler semantic similarity, that is based on the evaluation of the distance between the features of a service and the features requested by a user, in terms of the amount of edges that separates the two semantic concepts in the ontology.

[4] `http://www.semsorgrid4env.eu`

6 Conclusion

In this paper we have presented a semantic-based approach for retrieving services that refer to sensors installed in buildings. The approach has been validated and the results highlights the good performances of the algorithm in terms of precision and recall. Future work will focus on reducing the number of service invocations and optimizing the number of comparisons in the retrieval algorithm, in order to improve the response time performance and the scalability of the solution.

Acknowledgement

This work has been partially funded by Italian project "Industria 2015-Sensori" Grant agreeement n. 00029MI01/2011.

References

1. Chehri, A., Mouftah, H.T.: Service-oriented architecture for smart building energy management. In: Proc. of IEEE Int. Conf. on Communications. (2013) 4099–4103
2. Schreiber, F.A., Camplani, R., Fortunato, M., Marelli, M., Rota, G.: Perla: A language and middleware architecture for data management and integration in pervasive information systems. IEEE Trans. Software Eng. **38**(2) (2012) 478–496
3. Cappiello, C.: Data Quality and Multichannel Services. PhD thesis, Politecnico di Milano (2007)
4. Church, J., Motro, A.: Discovering service similarity by testing. In: Proc. of the IEEE Int. Conf. on Services Computing. SCC '11, Washington, DC, USA, IEEE Computer Society (2011) 733–734
5. Ye, J., Coyle, L., Dobson, S., Nixon, P.: Ontology-based models in pervasive computing systems. Knowl. Eng. Rev. **22**(4) (December 2007) 315–347
6. W3C: Review of Sensor and Observations Ontologies. `http://www.w3.org/2005/Incubator/ssn/wiki/Review_of_Sensor_and_Observations_Ontologies`
7. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with owls-mx. In: Proc. of the 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems, New York, NY, USA, ACM (2006) 915–922
8. Bianchini, D., Antonellis, V.D., Melchiori, M.: Capability matching and similarity reasoning in service discovery. In: In CAiSE Int. Workshop on Enterprise Modeling and Ontologies for Interoperability, EMOI 2005. (2005) 285–296
9. Günay, A., Yolum, P.: Structural and semantic similarity metrics for web service matchmaking. In: Proc. of the 8th Int. Conf. on E-commerce and Web Technologies. EC-Web'07, Berlin, Heidelberg, Springer-Verlag (2007) 129–138
10. Manikrao, U., Prabhakar, T.V.: Dynamic selection of web services with recommendation system. In: Int. Conf. on Next Generation Web Services Prac. (2005)
11. Li, Z., Bin, Z., Jun, N., Liping, H., Mingwei, Z.: An approach for web service qos prediction based on service using information. In: Int. Conf. on Service Sciences (ICSS). (2010) 324–328
12. Hau, J., Lee, W., Darlington, J.: A semantic similarity measure for semantic web services. In: Web Service Semantics Workshop at WWW. (2005)
13. Liu, M., Shen, W., Hao, Q., Yan, J.: An weighted ontology-based semantic similarity algorithm for web service. Expert Syst. Appl. **36**(10) (2009) 12480–12490