Using Colored Petri Nets for Verifying RESTful Service Composition

Lara Kallab, Michael Mrissa, Richard Chbeir, Pierre Bourreau

Using Colored Petri Nets for Verifying RESTful Service Composition

Lara Kallab[1,2], Michael Mrissa[1], Richard Chbeir[1], and Pierre Bourreau[2]

[1] Univ Pau & Pays Adour, LIUPPA, EA3000, France
{lara.kallab, michael.mrissa}@univ-pau.fr
rchbeir@acm.org
[2] NOBATEK/INEF4, Anglet, France
{lkallab, pbourreau}nobatek.com

Abstract. RESTful services are an attractive technology for designing and developing web-based applications, as they facilitate reuse, interoperability, and loosely coupled interaction with generic clients (typically web browsers). Building RESTful service composition has received much interest to satisfy complex user requirements. However, verifying the correctness of a composition remains a tedious task. In this paper, we present a formal approach based on Colored Petri Nets (CPNs) to verify RESTful service composition. First, we show how CPNs are utilized for modeling the behavior of resources and their composition. Then, we present how this formal model can be used to verify relevant composition behavior properties. Our solution is illustrated with a scenario built upon an energy management web framework developed within the HIT2GAP H2020 European project[3].

Keywords: REST Architecture · Resource Composition · Colored Petri Nets · Composition Properties Verification

1 Introduction

Nowadays, web-based applications provide functionalities as web services, to allow for language and platform independence and to improve interoperability with other services. The REST architectural style [4] has recently become the most adopted solution for designing and developing web services. This is due to i) its simplicity and ease of use that make services integration cost-effective, ii) its single approach that allows the use of generic clients to interact with any RESTful APIs, iii) its support for different data formats (e.g., XML and JSON), and iv) its ability to support caching for better performance and scalability. Hence, more and more, web-based applications provide functionalities as RESTful services that follow the principles of the REST architectural style, also referred to in this paper as resources. Each resource provides a well-defined functionality that meets a specific client request. However, answering some requests often requires the combination of two or more resources, forming a composition.

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

Although there have been many contributions related to RESTful service composition, ensuring the correctness of the composition behavior remains a challenge. In fact, verifying a composition usually relies on the formal verification of its behavioral properties [16] (e.g., Reachability, Liveness, and Persistence). Such verification typically depends on the formal modeling of the composition behavior via a modeling language with clear semantics. Several works have been carried out in this scope. Some RESTful composition approaches are based on formal languages (e.g., Petri Nets [1] [3], Finite State Machine (FSM) [18], and Process Algebra [15]), and others rely on services descriptions with embedded semantics such as in [14]. Although these approaches respect the majority of REST principles, they mainly contribute in modeling and constructing RESTful services composition without verifying its correct behavior.
In this paper, we propose a formal language based on Colored Petri Nets [9], known as CPNs, to model and verify RESTful service composition. The main contribution of this work is the mapping between CPNs model and RESTful services, to allow the use of CPNs behavioral properties for verifying RESTful service composition. Such verification is necessary, in order to avoid incorrect composition behavior and unecessary execution for an erroneous composition. CPNs were initially founded to provide design, analysis and verification techniques for concurrent processes behavior. They bring expressiveness in modeling complex systems, graphical visual notations that aid in following the process behavior simulation at each step, and more importantly, the ability to validate and verify the functional correctness of systems behavior [7]. Based on CPNs, our approach is able to ensure the correctness of a composition through the verification of the following behavior properties:

- **Interoperability**, checks if the resources involved into the composition can be linked together. This is related to data type compatibility between the linked resources where the output of a resource should be of the same type of the input of another resource.
- **Reachability**, is used to verify that the desired final composition state is reachable from the initial state.
- **Liveness**, ensures that all resources participating in the composition will be invoked during composition execution.
- **Persistence**, is used when parallel resources are executed simultaneously, and ensures that the occurrence of one resource will not disable another.

The remainder of this paper is organized as follows. Section 2 presents a scenario to motivate our work, and highlights the research problem we tackle. Section 3 gives a brief description on the principles of RESTful services and the basics of Colored Petri Nets (CPNs). Section 4 details our CPN-based approach for verifying RESTful service composition. Section 5 illustrates the proposed solution within our motivating scenario. Section 6 presents the related work and highlights the originality of our approach. Finally, Section 7 provides concluding remarks, and discusses perspectives for future work.

2 Motivating Scenario and Problem Statement

In this section, we present our motivating scenario illustrated in a web-based building energy management platform that is currently being developed within a H2020 European project called HIT2GAP.
Recent studies reveal that a huge gap exists between the design specifications and the operation of buildings energy performance[4]. This is due to various sources encountered in building life cycle phases (i.e., inaccuracy of the simulations tools, poor quality of the used construction equipment, inadequate verification tools, limited data analysis of the building in operation, etc.). With the aim of reducing the energy gap, HIT2GAP project provides a web-based energy management solution for managing the energy behavior of operational buildings. To do so, HIT2GAP uses mainly a 3 layered architecture as follows :

- **Field layer:** Includes heterogeneous data sensed from the building (e.g., internal temperature, energy consumptions, and other building related information such as doors and building levels), and from other sources (e.g., weather forecasts and occupants).
- **Core layer:** Contains 1) the HIT2GAP data repository to store the collected data, 2) a set of basic services (i.e., data preprocessing services used to correct erroneous data collected from the Field layer), and 3) web APIs through which third-parties modules access the framework services.
- **Management layer:** Contains third-parties modules (i.e., Forecasting, Fault Detection and Diagnosis, Occupants' Behavior Detection, etc.) developed by HIT2GAP partners.

HIT2GAP services and modules are encapsulated into RESTful services, called also resources. Each resource is dedicated to provide a specific functionality that satisfies a specific building actor request. However, some requests require the composition of several resources.

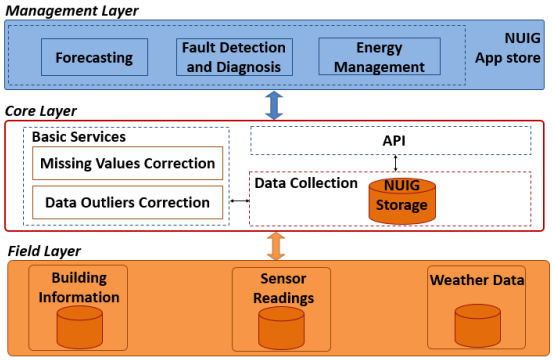


Fig. 1: HIT2GAP solution instance deployed in NUIG

To motivate our work, let us consider the case of the teaching and research building of the National University of Ireland Galway (NUIG), one of the HIT2GAP

[4] `https://www.designingbuildings.co.uk/wiki/Performance_gap_between_building_design_and_operation`

solution pilot sites. The building manager of NUIG wants to estimate the upcoming week heating energy consumption of the corresponding building. The prediction output will help him to anticipate building energy resource needs required for the resulted consumption, and analyze building energy behavior. To do so, the building manager will have to invoke several services simultaneously, embedded as resources, through the web interface of the HIT2GAP instance deployed for NUIG and illustrated in Figure 1. However, to satisfy his prediction demand properly, the building manager is ought to invoke the required resources and link them together correctly to reach the desired composition behavior. Figure 2 depicts the overall process to be executed for answering the building manager's request. It mainly requires the interaction with the following resources integrated into the HIT2GAP NUIG instance:

1. Data collection resources to collect data required for the prediction process:
   (a) Upcoming week predicted internal temperature, which is extracted directly from the HIT2GAP database
   (b) Upcoming week predicted external temperature, which is provided by an external weather forecast RESTful service
2. Data preprocessing resources that clean the collected data from the external weather forecast service:
   (a) Resource that manages and corrects outliers values, which are data values outside the range of most of the other values
   (b) Resource that manages and corrects empty or missing values retrieved during certain timestamps
3. Resource responsible for the prediction of the NUIG heat energy consumption. This resource, embedded into the Forecasting module, uses a prediction model considered already implemented in the module.

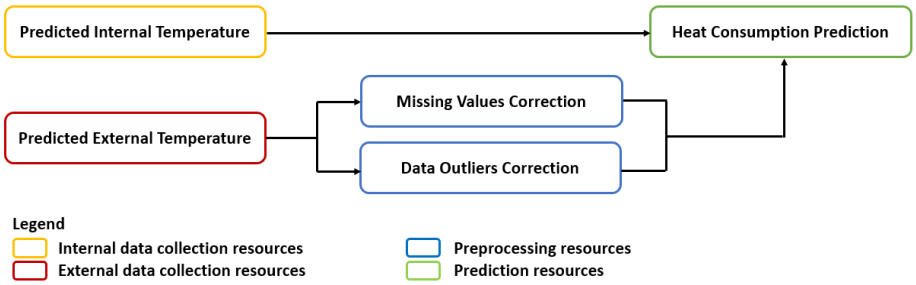


Fig. 2: NUIG resources involved in the prediction process

The scenario shows the resources composition needed to satisfy the request at hand. However, building the composition properly and ensuring its correct behavior is a difficult task for the building manager. In fact, several problems may occur when building and/or executing the composition:

- **Non-interoperability:** Links between the output of a resource and the input of another may be invalid. This is due to the difference of data types that each resource handles. For example, the resource responsible for correcting outliers values only processes an array of temperature values whose

data type is "Real". If it receives values from a previous resource with different data type (e.g., "Integer"), the composition will be erroneous. Such data type mismatch causes interoperability issues between resources.
- **Looping:** Starting the composition by collecting the required data (the internal and external predicted temperature), the process may not reach the final expected result, which is acquiring the predicted energy heat consumption of the NUIG building. This can be due to an end-loop occurred at a certain stage during composition execution, such as a loop in the preprocessing resources execution that can prevent the next resources to run.
- **Dead resources:** A resource, such as the external weather forecast, may not respond due to some technical problems on its server, and thus the next related resources involved into the composition process will not be invoked.
- **Conflict execution:** When dealing with parallel services such as preprocessing resources, a conflict in their execution may occur (deadlock).

To overcome these problems[5], we propose a formal language with clear semantics to model the behavior of RESTful services and verify the correctness of their composition. Such modeling language should cover the requirements below:

- **Support for REST principles:** Since HIT2GAP web services follow REST architecture principles, the formal language should be aligned with the main principles of REST architectural style. These principles are explained later in Section 3.1.
- **Data types handling:** The ability to handle the types of data flowing between services (i.e., String, Integer, etc.) allows composition syntax checking and thus a better management of the links between services.
- **Composition behavior verification:** In order to verify the correctness behavior of the composition, the modeling language should be able to verify the behavior properties considered important to HIT2GAP. These properties are: Reachability, Liveness, and Persistence.

In order to cope with the requirements mentioned above, we propose in this paper a formal language based on Colored Petri Nets (CPNs) to model Restful services and their composition. By using CPNs, our approach is able to handle data types and thus to check web services data compatibility. Moreover, with their formal syntax and semantics, they are able to validate the behavior of the built models through the execution of several verification properties embedded in open source and well known tools such as the CPN tools. The advantage of CPNs are detailed more in Section 3.2.

3 Background Knowledge

In this section, we provide the reader with the necessary information to reach a good understanding of the paper, through a brief reminder on the principles of REST and on Colored Petri Nets (CPNs).

[5] Please note that the service discovery and selection problems are out of the scope of this paper.

3.1 REST Principles

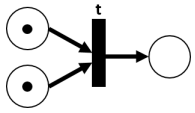
RESTful services, also called resources, are web services that follow the REpresentational State Transfer (REST) software architecture style [4]. A RESTful service provides a functionality through an abstract resource-oriented view identified by a Unique Resource Identifier (URI), and invoked via HTTP-based methods. We list below the main principles of RESTful services, inspired from [11]:

- **Resource Oriented and Addressability:** Resources are central elements addressable through their URI. They can be defined as objects with a type (e.g., JSON, XML, etc.), associated data (e.g., text files, images, etc.), relationships to other resources (i.e., Forecasting and Energy Management used in the HIT2GAP project), and a set of methods that operate on it (i.e., GET, POST, PUT and DELETE).
- **Uniform Interface:** Interacting with resources is realized through a uniform interface, which provides a set of standard operations (i.e., GET, POST, PUT and DELETE) implemented by the HTTP protocol. Each method has a well-defined semantics in terms of its effect on the state of the resource. It manipulates the resource state that can be transferred from/to clients, and represented in various types (e.g., JSON, XML, etc.).
- **Stateless Communication:** Every interaction with a resource is stateless. This means that each request is handled independently from the other, and request messages contain all the information that the server needs to process it. Stateless communication saves energy on the server side, as the state of the interaction with any client does not need to be stored in memory.
- **HATEOAS:** Known as Hypermedia As The Engine Of Application State, is a constraint of the REST paradigm used to provide directions to the client-agent regarding the next possible operations to be triggered. The principle is to include within returned server responses, the possible next resources URIs to follow, based on the current resource state. The methods used to invoke such resources can also be included.

3.2 Colored Petri Nets

Petri Nets are graphical and mathematical modeling language used to model distributed systems. They are designed to describe and study information processing systems, with concurrent, asynchronous, distributed, non-deterministic, parallel, and stochastic behaviors [10]. As shown in Figure 3, a Petri Net is graphically represented by a number of places (represented by circles) occupied by tokens (represented by black dots), and transitions (represented by rectangles). Transitions and places are connected via arcs. A transition may fire when each of its input places has at least one token (Figure 3(a)). When it fires, a token from each of its input places is removed, and a token is placed into every output place (Figure 3(b)). The number and position of tokens may change during the execution of the Petri Net transitions. The assignment of tokens to places designates a state or a marking of the net.

In ordinary Petri Nets, tokens cannot be distinguished and they are all identically represented as black dots. However, in more complex applications it is



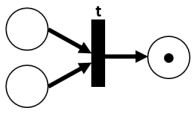


Fig. 3: Example of a single Petri Net

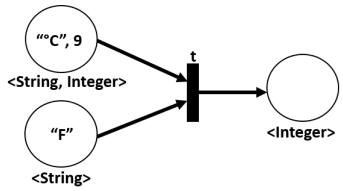


Fig. 4: Example of a Colored Petri Net

useful to allow the distinction between tokens and assign them some information. For these reasons, Colored Petri Nets (CPNs) combine the strengths of ordinary Petri Nets with the strengths of high-level programming languages [5], to allow handling data types and manipulating data values. In fact, within CPNs, each token can have a data type called a token color. Each color can be of a simple type (i.e., String, Integer, Boolean, etc.) or a complex type (i.e., array of String and Integer values). In addition, tokens with assigned colors can contain values. Normally, places in CPNs contain tokens of one type. An example of a CPN illustrating a temperature unit conversion is shown in Figure 4. As it is illustrated, the first input place holds a record type data containing 2 variables of String and Integer type respectively (one denoting the unit of measurement of the current temperature value, and the other the actual measured temperature value). The other input place holds a String type data representing the desired temperature unit of the Integer output place value.
Using CPNs makes our modeling approach independent from the data serialization format (i.e., XML, JSON, etc.), and has the potential to handle and show explicitly data types in the composition. Moreover, CPNs are able to analyze the modeled composition and investigate its performance by making simulations of the composition process and verify several behavioral properties (e.g., Reachability, Liveness and Persistence). These analysis can lead to important insights into the behavior of the composition and help in ensuring the correctness of its design. As such, the validation process for example can aid in preventing from having a dead transition that can block the execution of other transitions (Liveness), or guaranteeing that the desired final system state will be reached (Reachability). Formally, a CPN is defined as follows [13]:

Definition 1 - ***CPN*** *= (Σ, P, T, A, C, G, E, I), where:*

- *Σ is a finite set of non-empty types, called color sets*
- *$(P \cup T, A)$ forms a directed graph, where:*
  - *P (the set of places) and T (the set of transitions) are disjoint sets, such that $P \cap T = \emptyset$*
  - *$A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs, such that places are only connected to transitions, and vice versa*
- *$C : P \to \Sigma$ is the color function that maps places to elements of Σ*
- *$G : T \to \mathbb{B}$ associates a precondition g (a boolean expression) to each transition. g should be evaluated to true for T execution.*

– $E : A \rightarrow Expr$[6] *associates an expression* $E(a)$ *to each arc* a. $E(a)$ *is used to define input-output behavior of arcs, and may include variables such that:*
  - *Each variable in* $E(a)$ *has a type in* Σ
  - $\forall a \in A$, $C(E(a)) = C(p)$, *with* p *is the place connected to* a
– I *is the initialization function that maps each place* $p \in P$ *with an expression such that* $I(p)$ *is associated to the type* $C(p)$

4 CPN-based Approach for RESTful Service Composition

4.1 Resource Generic Interface

One of the key requirements of our modeling approach is to be aligned with REST principles. Therefore, it is essential to define the generic REST interface of a resource before presenting the interfaces of the resources involved in our motivating scenario. Generally, REST interface describes the required URI, HTTP method, query parameters[7], and responses of a resource. Responses includes:

– A list of the next resources to follow with the method used to invoke them. In this paper, the list can be empty when there are no resources to call.
– HTTP status code to indicate the query result. Such as, HTTP '200 OK' code denoting that the request has succeeded, and HTTP '201 Created' code designating that the request has been fulfilled and a new resource is created.
– The information provided by the resource, when it is available.

4.2 Interfaces of the Motivating Scenario Resources

In this paper, we restricted ourselves to the required interfaces in the motivating scenario, to ease the illustration of our approach. Table 1 lists the URIs of the composition scenario resources, and Table 2 defines their required interfaces.

Table 1: URIs of the prediction process resources

Id	URI
1	http://www.hit2gap.eu/predictions/pred-internal-temp
2	http://www.weatherforecast.com/forecast/external-temp
3	http://www.hit2gap.eu/missing-data-manager
4	http://www.hit2gap.eu/outliers-data-manager
5	http://www.hit2gap.eu/missing-data-corrected
6	http://www.hit2gap.eu/outliers-data-corrected
7	http://www.hit2gap.eu/predictions/pred-heat-consumption
8	http://www.hit2gap.eu/predictions/heat-consumption

URI 1 is called with GET to collect the predicted internal temperature according to 2 parameters: startdate and enddate, denoting the prediction time range requested by the building manager. The required data is retrieved directly from the HIT2GAP database, and considered as preprocessed data. The array 'PrInTemp' in the responses represents the predicted internal temperatures.

[6] Expr is a mathematical expression that will not be detailed further here due to space limitation

[7] Parameters are to be encoded in the URI or in the message body according to the HTTP format

URI 2 is invoked using GET to collect the predicted external temperature according to the same 2 parameters of the previous URI. The required data is retrieved from an external weather forecast resource. The array 'PrExTemp' represents the predicted external temperature. After data retrieval, URI 3 is invoked through POST to correct the missing values presented in the array 'PrExTemp', and URI 4 is called to correct the outliers values presented in the array 'PrExTemp'.
Through GET, URI 5 is called to retrieve the modifications applied on the predicted external temperature values obtained from the URI 3 (missing data manager). The '#dataset' represents the id of the preprocessed data, modified by URI 3. The array 'CorrMPrExTemp' contains the modifications applied on the predicted external temperatures.
Using GET, URI 6 is called to retrieve the modifications applied on the predicted external temperature values obtained from the URI 4 (outliers data manager). Similar to the previous step, the '#dataset' represents the id of the preprocessed data, modified by URI 4. The array 'CorrOPrExTemp' contains the modifications applied on the predicted external temperatures.
In our composition scenario, we considered that the merging of both URI 5 and URI 6 outputs, is being held on the client side to obtain the preprocessed external predicted temperatures array: [CorrExTemp{date, temp}].
URI 7 is invoked with POST to predict the energy heat consumption based on i) the startdate and the enddate, representing the prediction period range, and ii) the predicted internal temperatures with the preprocessed external temperature values previously collected. And finally URI, 8 is called using GET to retrieve the predicted heat energy consumptions obtained from the URI 7 and represented by '#dataset'. The array 'PrHeatEngCons' in the responses contains the predicted values of the heat energy consumption.

Table 2: Interfaces of the resources involved in the prediction process

URI	HTTP Verb	Parameters	Responses
1	GET	startdate=dd/mm/yyyy enddate=dd/mm/yyyy	200 OK [PrInTemp{date, temp}] {(POST, URI7)}
2	GET	startdate=dd/mm/yyyy enddate=dd/mm/yyyy	200 OK [PrExTemp{date, temp}] {(POST, URI3), (POST, URI4)}
3	POST	[PrExTemp{date, temp}]	201 Created {(GET, URI5)}
4	POST	[PrExTemp{date, temp}]	201 Created {(GET, URI6)}
5	GET	#dataset	200 OK CorrMPrExTemp{date, temp}] {(POST, URI7)}
6	GET	#dataset	200 OK [CorrOPrExTemp{date, temp}] {(POST, URI7)}
7	POST	startdate=dd/mm/yyyy enddate=dd/mm/yyyy [PrInTemp{date, temp}] [CorrExTemp{date, temp}]	201 Created {(GET, URI8)}
8	GET	#dataset	200 OK [PrHeatEngCons{date, HeatEngCons}] { }

4.3 Colored Petri Nets-based Formal Composition Model

As stated in Section 3.1, a resource can be invoked through HTTP methods to provide a specified functionality. It is published by a service provider, and located on a specific server. An exposed resource r, has a set of inputs, a set of outputs, and a function assigned to it. In our modeling approach, a resource can be atomic or composed. In the CPN model, we define i) an atomic resource as a single CPN with a single transition, input and output places, and ii) a composed resource as a set of linked CPN representing linked resources.
Before we formally define a resource, we define below the following sets:

- $\texttt{DataType} = \{\texttt{BasicT} \cup \texttt{ExtendedT}\}$ is the date types supported by a resource, such that:
  - $\texttt{BasicT} = \{\texttt{String}, \texttt{Integer}, \texttt{Real}, \texttt{Boolean}, \texttt{Date}, \text{etc.}\}$, denotes the basic data types known in programming languages
  - $\texttt{ExtendedT} = \{\texttt{Req}, \texttt{Status}\}$ denotes the extended data types defined to meet resources requirements
- $\texttt{Req} = (\texttt{HTTP}$ x $\texttt{U})$ is the set of HTTP requests sent to the URIs, where:
  - $\texttt{HTTP} = \texttt{POST}|\texttt{PUT}|\texttt{DELETE}|\texttt{GET}|\texttt{HEAD}|\texttt{PATCH}|\texttt{CONNECT}|\texttt{OPTIONS}|\texttt{TRACE}$ is the set of HTTP methods used to invoke the URIs
  - $\texttt{U}$ is a set of URIs based on the standard RFC3986 [8]
- $\texttt{Status} = (\text{Code, Desc})$ denoting the status of the resource response, where:
  - $\text{Code} \subseteq \mathbb{N}^*$ denotes the HTTP response status code
  - Desc represents the description of the HTTP code (e.g., 'Created', 'OK')

Definition 2 - *A RESTful resource r is defined as $r = (URI, N)$, where URI is the URI associated to r, and $N = (\Sigma, P, T, A, C, G, E, I)$ is a CPN such that:*

- $\Sigma \subseteq \texttt{DataType}$*, denoting the set of data types that the resource can process*
- *P is a finite set of input and output places of the resource, where:*
  - $P = P_{In} \cup P_{Out}$
  - $P_{In} = \bigcup_{i=1}^{\mathbb{N}^*} \{p_{in_i}\} \mid \bigcup_{n=1}^{\mathbb{N}^*} r_n.P_{In}$*, such that:*
    - $\bigcup_{i=1}^{\mathbb{N}^*} \{p_{in_i}\}$*, denotes the set of input places of an atomic resource. Each resource requires one input place, representing the request sent to it. Other input places can be defined according to the resources needs, such as the resources parameters.*
    - $\bigcup_{n=1}^{\mathbb{N}^*} r_n.P_{In}$*, denotes the set of input places of a composite resource*
  - $P_{Out} = \bigcup_{i=2}^{\mathbb{N}^*} \{p_{out_i}\} \mid \bigcup_{n=1}^{\mathbb{N}^*} r_n.P_{Out}$*, such that:*
    - $\bigcup_{i=2}^{\mathbb{N}^*} \{p_{out_i}\}$*, represents the set of output places of an atomic resource. Each resource requires two output places, denoting respectively the status response code and the set of the HTTP requests that can be sent to the next possible URI resources. Other places can be defined according to the resources needs, such as resources output results.*
    - $\bigcup_{n=1}^{\mathbb{N}^*} r_n.P_{Out}$*, denotes the set of output places of a composite resource*

[8] https://www.ietf.org/rfc/rfc3986.txt

- $T = t \mid \bigcup_{i=1}^{\mathbb{N}^*} r_i.T$. *t represents the functionality of an atomic r, whereas the union of T sub-resources represents the functionality of a composite resource.*
- *A is a finite set of arcs linking input places to transitions and transitions to output places, such that:* $P \cap T = P \cap A = T \cap A = \phi$
- *C is a color function. It associates a type from* Σ *to each place, where:*
  - $\exists p \in P_{In}$, *such that* $C(p) \in$ `Req`
  - $\exists p_1, p_2 \in P_{Out}$, *such that* $C(p_1) \in$ `Status`, *and* $C(p_2) \in$ `Req`
- *G is a guard function. It maps the transition* $t \in T$ *to a boolean guard expression g. The resource can only be executed if g is evaluated to true.*
- *E is an arc expression function. It maps each arc* $a \in A$ *into an expression that may include variables.*
- *I is an initialization function that associates places to initial values*

Based on our defined CPN formal model for RESTful service composition, we represent formally the composed resource resulted from the HIT2GAP prediction scenario depicted in Figure 5. Such formal language can be directly applicable to other scenarios and represent the corresponding web services as long as they are RESTful.

H2G_EnergyHeatPrediction = (URI, N), where URI is the address associated to the composition and $N = (\Sigma, P, T, A, C, G, E, I)$ is a CPN such that:

- $\Sigma = \bigcup_{i=1}^{8} r_i.\Sigma$ with r_i denoting the resources involved in the prediction process, and where:
  - $r_i.\Sigma \subseteq$ `DataType`, designates the data types handled by the resources participating into the composition
- $P = P_{In} \cup P_{Out}$, where:
  - $P_{In} = \bigcup_{i=1}^{8} r_i.P_{In}$, such that:
    - $r_1.P_{In} = r_2.P_{In} = r_3.P_{In} = r_4.P_{In} = \{p_{in1}, p_{in2}\}$, denoting that each of these resources has 2 inputs.
    - $r_5.P_{In} = r_6.P_{In} = r_8.P_{In} = \{p_{in1}\}$, denoting that each of these resources has 1 input.
    - $r_7.P_{In} = \{p_{in1}, p_{in2}, p_{in3}\}$, denoting that this resource has 3 inputs.
  - $P_{Out} = \bigcup_{i=1}^{8} r_i.P_{Out}$, such that:
    - $r_1.P_{Out} = r_2.P_{Out} = r_5.P_{Out} = r_6.P_{Out} = r_8.P_{Out} = \{p_{out1}, p_{out2}, p_{out3}\}$, denoting that each of these resources has 3 outputs.
    - $r_3.P_{Out} = r_4.P_{Out} = r_7.P_{Out} = \{p_{out1}, p_{out2}\}$, denoting that each of these resources has 2 outputs.
- $T = \bigcup_{i=1}^{8} r_i.T$, with $r_i.T =$ t denoting the specific functionality provided by each resource.
- A is a finite set of arcs linking input places to transitions and transitions to output places, such that: $P \cap T = P \cap A = T \cap A = \phi$.
- C is a color function. It associates a type from Σ to each place, where:
  - $\exists p \in P_{In}$, such that $C(p) \in Req$
  - $\exists p_1, p_2 \in P_{Out}$, such that $p_1 \neq p_2$, $C(p_1) \in Status$, and $C(p_2) \in Req$
- G is a guard function. It maps the transition $t \in T$ to a boolean guard expression g.

– E is an arc expression function. It maps each arc $a \in A$ into an expression that may include variables.
– I is an initialization function that associates places to initial values

We note that the composition URI is given after the verification of composition behavior that will be discussed in the following section.

4.4 Composition Properties

One of the main advantages of modeling RESTful services with CPNs format is the ability to analyze several behavioral properties of the composition. Mapping RESTful services and composition to CPNs with some extensions allows the execution of the algorithms related to CPNs properties. These algorithms that are used to check behavior properties in Colored Petri Nets still apply in our extended formal model, as it will be shown in this section.
To face the problems that may occur during composition execution, as it is presented in section 2, four properties have been considered important to verify in the HIT2GAP project: Reachability, Liveness, Persistence [10], and Interoperability. In the Reachability, Liveness, and Persistence definitions below, we use (N, M_0) to denote a Petri Net, N, with its initial Marking, M_0. The Petri Net marking, M, designates the state of the net, which corresponds to the assignment of tokens to places. The initial marking M_0 designates the availability of some data (tokens) in the input places of one or more resources involved in the composition, before launching the composition execution.

Definition 3 - ***Reachability*** - *A marking M_n is reachable from M_0 in a Petri Net N, if there exists a sequence of transitions firings from M_0 to M_n.*

One of the challenges in the composition process is to make sure that the final desired state is reachable from the initial state. In the prediction scenario, the desired final result is the predicted energy heat consumption, which is generated as an output from URI 8. To verify that the result is reached from the execution of data collection resources: URI 1 and URI 2, we use the Reachability graph.

Definition 4 - ***Reachability Graph (RG)*** - *It is a set of all the reachable markings of a Petri Net represented as nodes. The nodes are connected with arcs designating the firing of a transition.*

The Reachability graph algorithm, inspired from [10], is described as follows:

Algorithm 1 Reachability Graph

```
1: label the initial marking M0 as the root and tag it "new'
2: while "new" markings exists do
3:     select a new marking M
4:     if no transitions are enabled at M, tag M "dead-end'
5:     while there exist enabled transitions t at M do
6:         obtain the marking M' that results from firing t at M
7:         if M' does not appear in the graph, add M' and tag it "new'
8:         draw an arc with label t from M to M' (if not already present)
```

In our scenario, the Reachability property is true when: $\exists$ M_0 and $\exists$ $M \in$ RG as the end node, with M designating the final desired state.

Definition 5 - ***Liveness*** - *A Petri Net (N, M_0) is considered to be L_k-live if every transition t in the net is L_k-live. t is said to be:*

- *L_0-live, if it can never be fired in any firing sequence. In this case the transition t is considered deadlocked.*
- *L_1-live, if it can be fired at least once in some firing sequence*
- *L_2-live, if it can fire arbitrarily often*
- *L_3-live, if it can fire infinitely often*
- *L_4-live, if it may always fire*

Another challenge in our composition is to verify that the resources involved in the composition process will eventually be executed at least once. If for example, the preprocessing resources responsible of correcting erroneous data are not executed, the prediction resource will predict the building heat energy consumption based on inaccurate data. This will affect negatively the prediction results quality. Therefore L_1-live was our main focus, to ensure that all CPN transitions will eventually be fired by progressing through further allowed firing sequences. We note that transition firing depends on the availability of tokens (data) in all its input places. Thus, a resource can be executed only if its input places contain the required data which are: HTTP_VERB, URI, and some parameters (when it is necessary).
In our composition, a transition t is L_1-live when: $\exists\ M_0$ and $\forall\ t \in T\ in\ N$, $t \in$ RG. If not, t is considered dead.
By verifying both the Reachability and Liveness properties, we can be sure that the composition scenario contains no loop and all resources receive the required input in order to be executed.

Definition 6 - ***Persistence*** - *A Petri Net (N, M_0) is considered persistent if for any two enabled transitions the firing of one will not disable the other.*

The notion of Persistence is useful in the context of parallel resources and is related to conflict-free nets. It is important in our composition scenario to make sure that the execution of the parallel data preprocessing resources, will be done with no conflicts. With CPN, the Persistence property is automatically guaranteed. In fact, parallel resources execution is done in any order, and no execution will disable the other.
As for the Interoperability, by definition the CPN formalism put the following as a constraint: $\forall a \in A : [C(E(a)) = C(p)]$. This means that the CPN workflow execution will not be possible unless data flowing to and from a place are of the same type. We note that data flowing to a place correspond to a transition output, while data flowing from a place denotes the input of the next transition.

5 Experimental Illustration

We illustrate our proposed CPNs-based formal composition approach in our prediction scenario using the CPN tools[9], one of the most known tools for editing,

[9] `http://cpntools.org`

simulating, and analyzing Colored Petri Nets models. It provides a graphical user interface (GUI) with tool palettes and marking menus, to build the CPNs models. Moreover, it features syntax checking while the workflow is being constructed, and generates a standard state space report that contains information about behavioral properties of the modeled system.
Figure 5 represents the HIT2GAP scenario implemented according to our formalism. As it is illustrated, the model includes all the resources involved in the

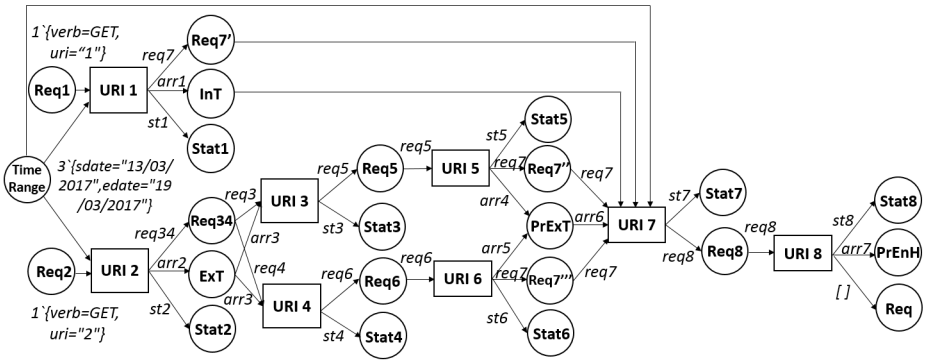
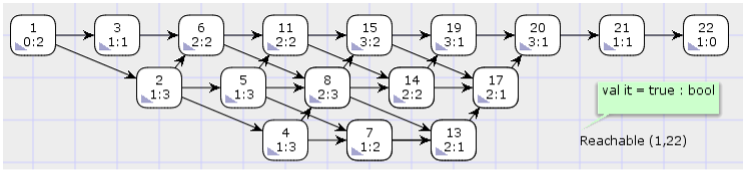


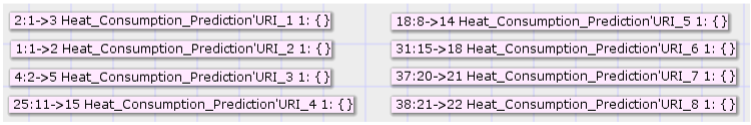
Fig. 5: CPN model for the resources composition relative to the prediction scenario

energy prediction process, with the required parameters and responses, described in Table 2. During our tests, we extended the input places related to URI 7 to respect the HATEOAS principle. In fact, due to the existing of several resources (URI 1, URI 5 and URI 6) that point out to URI 7 in their next resources to follow, we linked each of these resources to URI 7 transition.
Using the state space tool of the CPN tools, and by implementing queries via ML code (the functional programming language of the CPN tools), we were able to verify Reachability and Liveness properties. As for the Persistence and Interoperability properties, they were verified automatically during composition construction. Below are the properties tests applied to the composition scenario:

- **Reachability Test:** Figure 6(a) shows the Reachability graph of our scenario, containing a node for each reachable state. In total we have 22 states, with node 22 representing the final state that is the prediction output results (PrEnH). Moreover, we used the 'Reachable (1,22)' boolean function (written in ML code) to test if state 22 is reached from state 1. The returned boolean value equal to "true" verifies the Reachability property.
- **Liveness Test:** Liveness property is verified through analyzing the Reachability graph arcs, which are labeled by the resources responsible of the state changing. Figure 6(b) shows examples of some Reachability graph arcs labels, appeared when clicking on the arcs. It proves that all URIs (from 1 to



(a) Reachability graph of the prediction process



(b) Examples of the Reachability graph arcs labels

Fig. 6: Reachability graph

8) are executed at least once. Moreover, and when generating the state space report through the state space tool of the CPN tools, several information can be retrieved including Liveness property results. Figure 7(a) for example shows the statistics representing the number of nodes and arcs of the composition scenario, and Figure 7(b) proves that there are no dead transition instances, denoting that all the resources will be eventually executed starting from the initial state.

```
Statistics
--------------------------

State Space
   Nodes:  22
   Arcs:   38
   Secs:   0
   Status: Full
```

(a) State space report statistics

```
Liveness Properties
------------------------------

Dead Transition Instances
   None
```

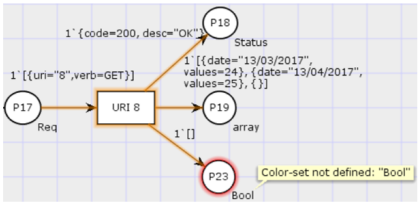
(b) Dead transitions

Fig. 7: Information retrieved from the state space report

- **Persistence Test:** The persistence property is guaranteed automatically in the CPN tools, due to the CPN ability in modeling parallel systems. In fact, the execution of parallel transitions can be done without interruptions, as long as each transition has the required data in all of its input places.

```
▼Standard declarations
  ▼colset HTTP_VERB = with GET | POST | PUT | DELETE;
  ▼colset URI = string;
  ▼colset Request = record verb:HTTP_VERB * uri:URI;
  ▼colset Req = list Request;
  ▶colset date
  ▼colset Range = record sdate:date* edate:date;
  ▶colset INT
  ▼colset arr = record date:date * values:INT;
  ▶colset STRING
  ▼colset Status = record code:INT * desc:STRING;
```

(a) Color sets used to model the required data types



(b) Interoperability Issue

Fig. 8: Defined color sets and Interoperability issue

- **Interoperability Test:** In order to represent the flowing data types between resources, we defined the color sets as shown in Figure 8(a). Using CPNs, our approach allows to verify the Interoperability property between the linked resources. CPN tools check the syntax of the nets during their construction where errors, such as in data types, can be visually seen through specific color indications, as shown in Figure 8(b).

6 Related Work

6.1 Petri Net-based Approaches

Formal languages are defined in [3] [1] to describe RESTful services based on high-level Petri Nets. However, these approaches ignore internal hypermedia links, describe all tokens in XML only, and do not verify the correctness of composition behavior. In our work, we explicitly define data types as colors, within Σ that contains standard types and other defined types required for the resources. Also, in our approach we propose a unique definition of a resource as a service, identified by a URI, which can be invoked to provide its functionality. On top of that, we focus on how CPNs properties can help verify the correctness of RESTful composition behavior.

6.2 FSM-based Approaches

In [18], RESTful systems are modeled through a non-determinitic Finite State Machine (FSM) approach with epsilon transitions that do not need to read an input symbol in order to modify the system's state. The proposed model follows some REST design principles, i.e., uniform interface, stateless client-server operation, and code-on-demand execution. The main advantage of this work is that it supports hypermedia links between internal resources of a single web service. However, composition between different RESTful services are not modeled. Moreover, there are no composition verification properties used to ensure the correct behavior of the composition execution.

6.3 Linear Logic-based Approaches

In [17], Intuitionistic Linear Logic (ILL) is used to model formally RESTful web services. The main contribution of this approach is web services composition modeling, and the ability to ensure composition completeness and correctness, through theorems based on propositional Linear Logic and π-calculus. However, although it respects the main principles of REST architectural style principles, linear Logic is a complicated formal language that requires extra efforts from web engineers to put it in practice.

6.4 Process Algebra-based Approaches

In [15], RESTful services are described through the combination of process calculi format with tuple space computing, a model for managing a distributed object system. Based on this work, a semantic RESTful resource is formalized as a process associated with a triple space and a URI used for handling remote

requests. However, the proposed approach does not support HATEAOS principle, nor even verification properties to check the correctness of the composition.

6.5 Semantic-based Approaches

RESTdesc [14], is a solution that describes services' functionalities through the Notation3 syntax with embedded semantics. Although it supports the automatic web services discovery and composition by constructing a graph that links and identifies resources, it does not verify the correctness of the composition.

6.6 SOAP-based Approaches

Before REST technology has been emerged, numerous work were conducted to compose services based on SOAP protocol [12]. In [6,8,2] SOAP web services have been translated to Petri Nets-based models. Apart from being SOAP-oriented services, none of them handle data types and they lack in analyzing composition behavior properties.

7 Conclusion

In this paper, we proposed a Colored Petri Nets-based approach for composing RESTful services, in the context of a web-based energy management solution developed in the HIT2GAP European project. We first exposed our CPN-based formalism to model the behavior of REST resources with their composition. And then, we showed how the verification of composition behavior properties (i.e., Interoperability, Reachability, Liveness, and Persistence) can be done based on our formal defined model.
Future work includes studying the limitations of our approach when the composition involves a large number of services. As well, quality of service constraints could be also modeled and taken into consideration in the verification. We also intend to extend resources descriptions with semantic annotations to automate the service orchestration task.

Acknowledgement

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. Alarcon, R., Wilde, E., Bellido, J.: Hypermedia-driven restful service composition. In: International Conference on Service-Oriented Computing. pp. 111–120. Springer (2010)
2. Chemaa, S., Elmansouri, R., Chaoui, A.: Web services modeling and composition approach using object-oriented petri nets. arXiv preprint arXiv:1304.2080 (2013)

3. Decker, G., Lüders, A., Overdick, H., Schlichting, K., Weske, M.: Restful petri net execution. In: Bruni, R., Wolf, K. (eds.) Web Services and Formal Methods, 5th International Workshop, WS-FM 2008, Milan, Italy, September 4-5, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5387, pp. 73–87. Springer (2008), http://dx.doi.org/10.1007/978-3-642-01364-5_5
4. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000)
5. Gehlot, V., Nigro, C.: An introduction to systems modeling and simulation with colored petri nets. In: Proceedings of the Winter Simulation Conference. pp. 104–118. Winter Simulation Conference (2010)
6. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: Proceedings of the 14th Australasian database conference-Volume 17. pp. 191–200. Australian Computer Society, Inc. (2003)
7. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured petri nets. International Journal on Software Tools for Technology Transfer (STTT) 2(2), 98–132 (1998)
8. Li, B., Xu, Y., Wu, J., Zhu, J.: A petri-net and qos based model for automatic web service composition. JSW 7(1), 149–155 (2012)
9. Liu, D., Wang, J., Chan, S.C., Sun, J., Zhang, L.: Modeling workflow processes with colored petri nets. computers in industry 49(3), 267–281 (2002)
10. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
11. Pautasso, C.: Restful web services: principles, patterns, emerging technologies. In: Web Services Foundations, pp. 31–51. Springer (2014)
12. Suda, B.: Soap web services. Retrieved June 29, 2010 (2003)
13. Tian, B., Gu, Y.: Formal modeling and verification for web service composition. JSW 8(11), 2733–2737 (2013)
14. Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., Van de Walle, R., Vallés, J.G.: Description and interaction of restful services for automatic discovery and execution. In: 2011 FTRA International workshop on Advanced Future Multimedia Services (AFMS 2011). Future Technology Research Association International (FTRA) (2011)
15. Wu, X., Zhu, H.: Formalization and analysis of the REST architecture from the process algebra perspective. Future Generation Comp. Syst. 56, 153–168 (2016), http://dx.doi.org/10.1016/j.future.2015.09.007
16. Yang, Y., Tan, Q., Xiao, Y.: Verifying web services composition based on hierarchical colored petri nets. In: Proceedings of the first international workshop on Interoperability of heterogeneous information systems. pp. 47–54. ACM (2005)
17. Zhao, X.: A linear logic approach to RESTful web service modelling and composition. Ph.D. thesis, University of Bedfordshire, UK (2013), http://hdl.handle.net/10547/301103
18. Zuzak, I., Budiselic, I., Delac, G.: A finite-state machine approach for modeling and analyzing restful systems. J. Web Eng. 10(4), 353–390 (2011), http://www.rintonpress.com/xjwe10/jwe-10-4/353-390.pdf