We consider a set S of services, hosted by a set N of nodes of a distributed system (e.g. nodes of an edge cloud architecture). We assume that each node consumes energy (to maintain its state when activated and to support the activities of software-implemented services it is hosting), and can possibly produce green energy (e.g. by wind or solar sources) that can be used to totally or partially compensate its local energy consumption. Each node offers to services it is hosting a set of basic services needed for their execution (e.g., computing, communication and storage services). For the sake of simplicity, in this paper we limit ourselves to consider only computing and communication services offered by a node. Moreover, we also assume that each node offers a single type of computing and communication service. We leave to future work the extension of our model to other basic services categories (e.g.: storage) and to different types of services within each category (e.g., both general purpose CPU and GPU as computing services).

In the model described below, we denote by S, S_i single elements of the set S, and by n, n_i single elements of the set N. We denote by $compserv(n) \in S$ and $commserv(n) \in S$ the computing and communication service offered by a node $n \in N$, respectively, and by $B \subseteq S$ the set of all basic computing and communication services offered by nodes in N (i.e., $B = \{S \in S | S = compserv(n) \vee S = commserv(n)$ for some $n \in N\}$). Finally, we denote by $node(S) \in N$ the node hosting service S. With these definitions, it always holds $node(compserv(n)) = n$ and $node(commserv(n)) = n$.
where: itemize

e ach $\sigma_{S,d} \in \Sigma_S$ models a traffic source, represented by the rate of service requests that terminal service

$(commserv(n)) = n$.

Given these initial definitions, we model a service $S \in S$ as a tuple $\langle Type, Deps, Prov, Req, L, Q, E \rangle$, where:

itemize

[illegible]mod.

itemize

A service is either fully resolved or partially resolved. A service S is fully resolved if either: (i) S has no dependencies ($S = \emptyset$? $S \in B$?); or (ii) for all $d \in S$ there exists a fully resolved service $S' \in S.Prov$ such that $match(d, S') > 0 \wedge node(comp(S)) = node(comm(S))$ (i.e., the computing and communication services resolving the *comp* and *comm* dependencies of S are colocated on the same node).

On the other hand, a partially resolved service S has a non-empty list of dependencies, and at least one dependency is either not matched, or is matched by a partially resolved service.

spero di non aver sbagliato con il senso delle frecce e Prov e Req :-) ; controllare A service assembly A is a directed graph $A = (S, E)$, where $E \subseteq S \times S$ is the set of resolved dependencies. Specifically, a directed edge $(S_i, S_j) \in E$ denotes that S_i is using S_j to resolve one of its dependencies. In general, a given S_i has multiple simultaneous outgoing bindings (towards services in $S_i.Prov$), one for each dependency, and can have multiple simultaneous incoming bindings from other services (belonging to $S_i.Req$), using S_i to resolve one of their dependencies.

According to this model, the problem introduced above can be restated as: framed how to dynamically build and maintain over time a fully resolved assembly of distributed services that are blue able to collectively fulfill functional, QoS and energy requirements in case of openness, variability, unpredictability and scalability of the execution environment.

framed

Load model sec:loadmodel As pointed out in the Introduction, we want to explicitly take into account in our QoS&energy-driven assembly construction, the possible load-dependent nature of both the QoS and energy consumption of each service. Indeed, with regard to QoS attributes, load dependence obviously holds for attributes in the performance domain (e.g., response time), where the load has a negative impact on their value. Besides, it may also hold for other domains like dependability, where increasing load could increase the likelihood of failures SoftwareAging2007,Schroeder:2006, or cost, for example in case of cost schemes based on congestion pricing congPricing. With regard to energy consumption, it is reasonable to assume that it increases with increasing load addressed to a service.

To this end, we assume that each service S includes in its characterization a load profile $S.L$ that allows capturing in a simple yet sufficiently expressive way the load dependencies among services in the system.

For each service $S \in S$ the associated load profile is defined as follows:

footnotesize equation S.L = cases $\Sigma_S = \{\sigma_{S,d} \in R^+ | d \in S\} if$S$\in U$

S addresses to its dependency d; Forse userei Source Service, Terminal mi da l'idea di un servizio senza dipendenze

e ach $\theta_{S,d} \in \Theta_S$ is a transfer function, where $\theta_{S,d}(\lambda)$ is the average rate of service requests sent to dependency d by a non-terminal service S when S is subject to an incoming rate λ of service requests.