PAPER Special Section on Future Directions of Research and Development on Communication Quality

# An Intent-Based System Configuration Design for IT/NW Services with Functional and Quantitative Constraints

**Takuya KUWAHARA**[†a)], *Nonmember*, **Takayuki KURODA**[†b)], *Member*, **Takao OSAKI**[†c)], *Nonmember*, *and* **Kozo SATODA**[†d)], *Member*

**SUMMARY** Network service providers need to appropriately design systems and carefully configuring the settings and parameters to ensure that the systems keep running consistently and deliver the desired services. This can be a heavy and error-prone task. Intent-based system design methods have been developed to help with such tasks. These methods receive service-level requirements and generate service configurations to fulfill the given requirements. One such method is search-based system design, which can flexibly generate systems of various architectures. However, it has difficulty dealing with constraints on the quantitative parameters of systems, e.g., disk volume, RAM size, and QoS. To deal with practical cases, intent-based system design engines need to be able to handle quantitative parameters and constraints. In this work, we propose a new intent-based system design method based on search-based design that augments search states with quantitative constraints. Our method can generate a system that meets both functional and quantitative service requirements by combining a search-based design method with constraint checking. Experimental results show that our method can automatically generate a system that fulfills all given requirements within a reasonable computation time.

***key words:*** *intent-based system configuration, automated system design, design space exploration, quantitative requirement*

## 1. Introduction

As network services become more complex, network service providers are faced with a heavier burden in terms of deployment and management of the underlying systems. This mainly stems from the detailed and complex requirements of the underlying system/network architecture. Service providers need to finely adjust the components of their systems to ensure consistent cooperative operations, even though their service-level requirements might be simple. In practical cases, this creates a heavy burden on the providers as they need to manually organize the systems to fulfill the service requirements, dependencies, and constraints derived from the system components.

To address this issue, researchers have focused on ways to generate a networked system architecture based on the service requirements. One practical approach is *template-based* design [1]–[3], where the service requirements are given as the parameters on a prepared system template and then template-based design engines implement them on an appropriately adjusted system architecture based on the template. While template-based approaches can deploy the desired services easily and quickly, they lack flexibility when it comes to the system architecture. Manual adjustment is required if the desired system components are not available on the templates or if the system architecture deviates from the templates, which drives up the labor and time costs.

To enable more flexible design method, some model-driven developments (MDDs) have adopted a more flexible framework to design systems. MODACLOUDS [4] provides system developers with a semi-automatic system design method for cloud computing environments by using abstract service models and a decision support system (DSS) to analyze service models. SASSY [15] can generate a system configuration from a service-level architecture described in their visual specification language called service activity schema (SAS). While these methods enable a more flexible design of the system architecture than template-based methods, they require information about the desired service to determine a concrete system configuration and do not *explore* a suitable system design when it is unclear how a given service requirement can be met.

Search-based design [5], [10], [12] is one approach to accept ambiguous service requirements as input and flexibly design system architectures. Systems and service requirements can be represented by a graph model (called *a topology* [12]) that visualizes the system components and the relationships between them. Unlike a model of systems, a model of service requirements contains abstract parts. A search-based design engine concretizes the abstract parts of a service requirements model by utilizing domain-specific knowledge and gradually transforms it into a model of completely concrete system configurations.

As mentioned above, the search-based design approach collects and uses domain-specific knowledge of domain experts as a model to translate abstract requirements into a system. For example, consider developers of a certain software "$SW_A$". They are domain experts on software A. By describing the input devices, dependent packages, DBs, etc. that are necessary for the operation of software $SW_A$, they can define the necessary conditions for its operating environment. Suppose that software $SW_A$ needs to receive input from a device through API. The developers can define more specific ways to achieve this, such as setting up API endpoints and adding an HTTP communication-enabled en-

Manuscript received September 1, 2020.
Manuscript revised November 27, 2020.
Manuscript publicized February 4, 2021.

[†]The authors are with NEC Corporation, Tokyo, 108-8001 Japan.
a) E-mail: kwhr111011@nec.com
b) E-mail: kuroda@nec.com
c) E-mail: t-osaki@nec.com
d) E-mail: satoda@nec.com

DOI: 10.1587/transcom.2020CQP0009

vironment. By using such domain-specific knowledge, a search-based design engine can remove the abstract parts of the service requirements model pertaining to the usage and functionality of software $SW_A$ and transform it into a more concrete model. In [12], the rules for this transformation are called *refinement rules*. By repeatedly applying appropriate domain-specific knowledge, a search-based design engine can concretize all the abstract parts of a given service requirement and obtain a completely concrete system configuration.

Unlike template-based designs, which require the preparation of a model of the entire system, the search-based design collects domain-specific knowledge models. For example, the definition of a model for software $SW_A$ includes information about which components are required for its operation, but does not include information about the operating environment and the other requirements of those components. As mentioned above, the search-based design domain-specific knowledge model does not restrict the entire applicable service requirements to a specific form, as it concretizes only those requirements that are focused on the target domain. The above mechanism enables search-based design to offer a flexible design that matches diverse service requirements, which is difficult to do with template-based automated design.

As we mentioned above, a service developer can identify the desired system configuration that satisfies the *functional* service requirements by using search-based design. However, developers typically also have a lot of *quantitative* service requirements that specify the limitation of cost, upper-bound of available RAM, data-handling capacity and so on. One effective approach to address quantitative requirements is *constraint-based system design*, which obtains a suitable system architecture by solving a constrained optimization problem on the numerical parameters of systems. Constraint-based design is a domain-specific method such as QoS-aware composition of Web services [13], optimized service placement [14], and optimized VNF-node placement [18] and is not a generic system design method.

As opposed to constraint-based design, it is difficult for existing search-based methods to deal with quantitative requirements. They can handle the functional requirements, which can be tested simply by checking a *local* part of the systems, and thus enable complete concretization by repeatedly rewriting each abstract part. In contrast, the quantitative requirement of a system often requires a *global* viewpoint when testing its consistency.

For example, consider a case where a search-based design engine is going to deploy two services in the same infrastructure. Even if these two services are unrelated in terms of functionality, they share network/computational resources in the infrastructure and so their available resources are in a trade-off relationship. However, in existing search-based design methods, these two systems are independently concretized due to their functional irrelevancy, so the engine cannot balance the two amounts of resource usage and may output systems that do not satisfy the quantitative constraints.

In this paper, we propose a novel search-based system design method that can deal with both the functional and quantitative service requirements at the same time. We extend the existing search-based design by associating topologies and refinement rules with additional quantitative constraints, called *constrained topologies* and *constrained refinement rules*, respectively. To refine a constrained topology by a constrained refinement rule, our design engine associates the topology with the quantitative constraints of the rule, performs consistency checking of the quantitative constraints of the refined topology, and accepts the topology as a search target only if the constraint is found to be consistent. This technique enables our method to proceed with the search process while always associating a global quantitative constraint with each topology. As a result, our search-based engine searches only topologies that satisfy all the quantitative constraints necessary for the normal operation of systems and generates a system configuration that satisfies both the functional and quantitative requirements.

This paper describes the theoretical methodology of our search-based design method and reports the results of evaluations using a prototype. In Sect. 2, we formalize the fundamental framework of search-based design, and in Sect. 3, we illustrate our motivating example. Section 4 describes our search-based design method. In Sect. 5, we report our case studies and the results. We conclude in Sect. 6 with a brief summary and mention of future work.

## 2. Search-Based System Design

Our method is based on *Weaver*, the search-based system designer proposed by Kuroda et al. [12]. In this section, we briefly describe Weaver's data format and algorithm.

### 2.1 Overview

The overview of Weaver is shown in Fig. 1. Graphs in the rectangular boxes are topologies and the arrows between them are refinements. Weaver deals with systems including abstract parts called topologies (described in Sect. 1). First,

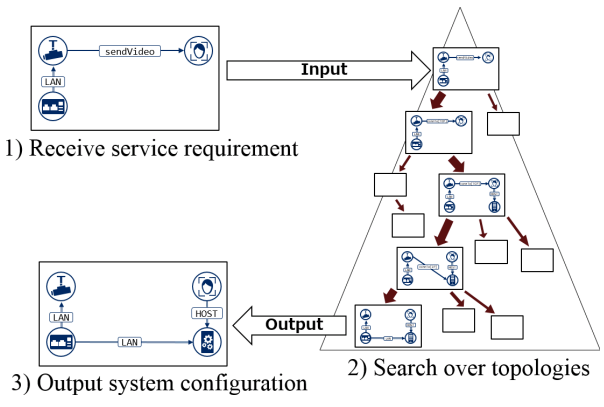


**Fig. 1** Overview of weaver [12].

it receives a customer's service requirement in the format of a topology. Second, the given service requirement is repeatedly converted by applying *refinements* until all abstract parts in the service requirement are concretized and a completely concrete topology is obtained. This refinement procedure is performed by creating *a search tree* whose root node is the service requirement. Finally, Weaver outputs the obtained completely concrete topology as a system configuration.

### 2.2 Data Format

In this section, we define the data formats used in Weaver.

#### 2.2.1 Types and Topology

A topology is a fundamental *graph* format representing systems. Its design was inspired by OASIS TOSCA [17]. While a system specification of TOSCA represents a concrete system configuration, the topology data format gives us a unified representation for service requirements, service configurations, and *partially refined* service requirements. A topology consists of components and relationships as defined below.

First, we introduce component types and relationship types. A **relationship type** is simply defined as an identifier. A **component type** is defined as a pair $v = (name, req)$. The *name* is an identifier specifying $v$ and the *req* is a set of relationship types, called *requirement fields* of the type. Each rtype $\in$ *req* means that the component $v$ requires that there is exactly one relationship $v$ –(rtype)→ •. When it is clear from the context, we simply refer to the component and relationship types as "types".

The component types are divided into *abstract component types* and *concrete component types*. Concrete component types correspond to concrete system components such as actual hardware and software. Abstract component types correspond to abstract components such as "some application" and "some server". The relationship types are also divided into *abstract relationship types* and *concrete relationship types*.

The *deriving relation* "$t_1 \prec t_2$" is defined between two component types, which intuitively means $t_1$ is a *specialized* type of $t_2$. In addition, we define a binary relation $t_1 \preceq t_2$ that means $t_1$ is derived from or equal to $t_2$.

A **component** "$v$ : ctype" is defined by an identifier $v$ and a component type ctype. A **relationship** "$v_{src}$ –(rtype)→ $v_{dst}$" is defined by two components $v_{src}, v_{dst}$ and a relationship type rtype. When components and relationships have a concrete type, they are said to be *concrete*. Otherwise, they are said to be *abstract*.

Now, we can formalize a topology as follows. A **topology** is defined as a pair $t = (V, E)$, where $V$ is a set of components and $E$ is a set of relationships on $V$. A topology represents a structure of services such as TOSCA in the form of a directed typed graph. We denote a set of all components in a topology $t$ by $V(t)$ and a set of all relationships in $t$ by $E(t)$.

### 2.3 Refinement

A refinement is a procedure to convert topologies by applying *refinement rules*. A **refinement rule** is defined as a pair of topologies $r = (t_{lhs}, t_{rhs})$ where $t_{lhs}$ and $t_{rhs}$ are called the *left-hand side* and *right-hand side* of $r$, respectively. A refinement rule $r = (t_{lhs}, t_{rhs})$ needs to satisfy the following conditions; (1) When $t_{lhs}$ has a component $v$ : $\text{ctype}_1$, $t_{rhs}$ has a component $v$ : $\text{ctype}_2$ such that $\text{ctype}_2 \preceq \text{ctype}_1$ holds, and (2) all identifiers of components are *placeholders* {1}, {2}, ..., {n}. Here, we define the *size* of $r$ as $n$.

A **matching** is a mapping from placeholders {1}, {2}, ..., {n} to identifiers. A matching is defined as a sequence of identifiers $m = id_{\{1\}}, id_{\{2\}}, \ldots, id_{\{n\}}$ and means a mapping defined as $m(\{k\}) = id_{\{k\}}$.

We can define an **action** $r[m]$ by pairing a refinement rule $r = (t_{lhs}, t_{rhs})$ and a match $m$. The action $r[m]$ is said to be **applicable** to a topology $t$ if and only if the following conditions are fulfilled: (1) The size of $r$ is equal to the length of $m$. (2) For all components $(\{k\}, \text{ctype}_r) \in V(t_{rhs})$, if a component $(\{k\}, \text{ctype}_l)$ is in $V(t_{lhs})$, then $(m(\{k\}), \text{ctype}_t)$ is in $V(t)$ such that $\text{ctype}_t \preceq \text{ctype}_l$ holds and $\text{ctype}_r \preceq \text{ctype}_t$ or $\text{ctype}_t \preceq \text{ctype}_r$ holds. (3) For all components $(\{k\}, \text{ctype}_r) \in V(t_{rhs})$, if a component $(\{k\}, \text{ctype}_l)$ is not in $V(t_{lhs})$, then a component with the identifier $m(\{k\})$ is not in $V(t)$. (4) For all relationships {i}–(rtype)→{j} $\in E(t_{lhs})$, a relationship m({i}) –(rtype)→ m({j}) is in $E(t)$.

When $r[m]$ is applicable to $t$, we can define a *refinement* process as the following four steps; (1) Remove a relationship m({i}) –(rtype)→ m({j}) from $E(t)$ if {i}–(rtype)→{j} $\notin E(t_{rhs})$, (2) add a new component $(m(\{i\}), \text{ctype})$ to $V(t)$ if $(\{i\}, \text{ctype}) \in V(t_{rhs}) \setminus V(t_{lhs})$, (3) add a new relationship m({i}) –(rtype)→ m({j}) to $E(t)$, if {i}–(rtype)→{j} $\in E(t_{rhs}) \setminus E(t_{lhs})$, and (4) modify the type of a component $(m(\{i\}), \text{ctype}) \in V(t)$ to $\text{ctype}_r$ if $\text{ctype}_r \prec \text{ctype}$ and $(\{i\}, \text{ctype}_r) \in V(t_{rhs})$.

We denote the converted $t$ by the action $r[m]$ by $r[m](t)$.

**Example 1.** *Fig. 2 shows an example of the refinement where topology $t_1$ is transformed into topology $t_2$ by rule* ***SENDVIDEO*** *and match $m_{\text{expl}}$ =* `camera, vs`*. We assume that deriving relation* `VideoSurveillance` $\preceq$ `App` *holds and so node* `{2}` *of type* `App` *corresponds to* `vs` *of type* `VideoSurveillance`*.*

*Because rule* ***SENDVIDEO*** *means that relationship* `sendVideo` *between* `{1}` *and* `{2}` *can be replaced by* `connectTo[HTTP]`*,* `sendVideo` *between* `camera` *and* `vs` *in topology $t_1$ is replaced by* `connectTo[HTTP]`*.*

### 2.4 System Configuration

As stated in Sect. 2.1, the goal of search-based design is to generate a *system configuration* for a given service requirement. Formally, a system configuration is defined as

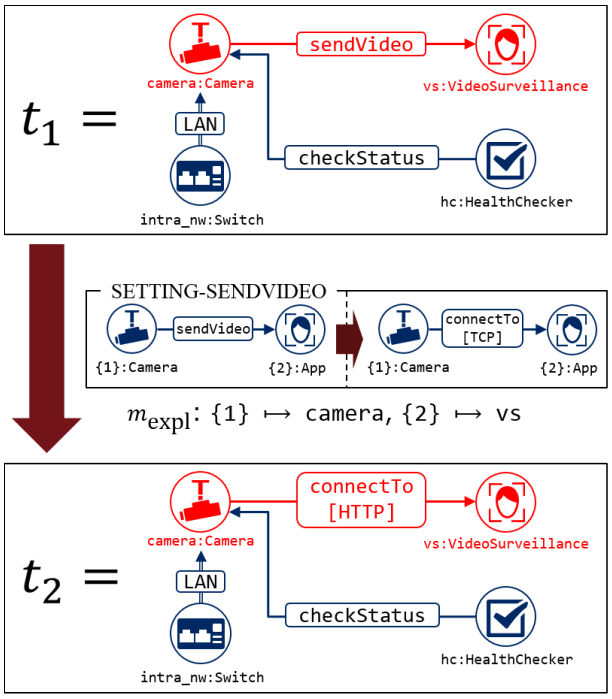


**Fig. 2** Refinement of the topology $t_1$.

a *completely concrete* topology. A topology $t$ is said to be **completely concrete** when the following conditions are fulfilled: (1) All $v \in V(t)$ and all $e \in E(t)$ are concrete. (2) For all $v = (id_v, req_v) \in V(t)$, if rtype is in $req_v$, then there is exactly one component $v' \in V(t)$ and relationship $v \xrightarrow{\text{rtype}} v' \in E(t)$.

### 2.5 Search-Based Design Algorithm

As shown in Fig. 1, Weaver performs a tree search with topologies as nodes and refinements as edges. Topologies occurred in the process of tree search are called *search candidates*. Weaver chooses one search candidate and repeats operation of adding the results of applying applicable refinements to the search candidate to the search tree to discover a system configuration satisfying a given service requirements.

## 3. Motivating Example

Our motivating example consists of services for video surveillance and health checking for cameras. This example is also used as the running example in Sect. 4. We assume that a customer has network cameras and wants to install a new video surveillance system for security. For simplicity, the customer has only one camera.

Figure 3 shows all component types used in our motivating example and deriving relations between them. As shown in Fig. 3, component types are represented as icons. Each balloon associated with a component type indicates requirement fields of the type. For example, the type `App` is defined as a pair (`App`, {`HOST`}).

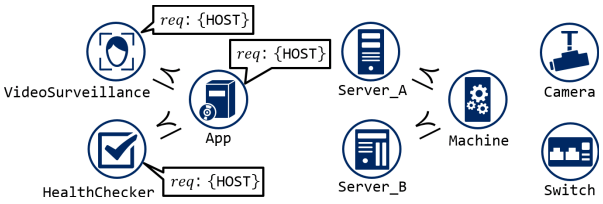


**Fig. 3** Component types used in our motivating example.

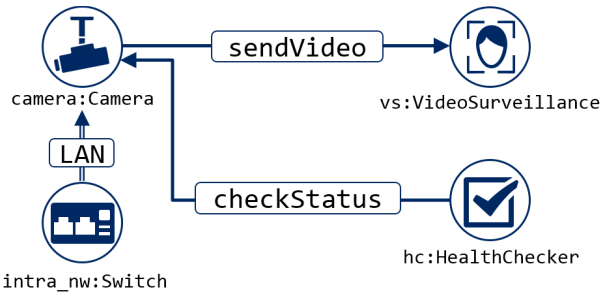


**Fig. 4** $t_{\text{M.in}}$: a service requirement of our motivating example.

The functional requirement of our motivating example is as follows. Its topology $t_{\text{M.in}}$ is shown in Fig. 4. Each circle in Fig. 4 represents a component in $t_{\text{M.in}}$, contains an icon that represents the component's type and is labeled by its identifier and type. Each double-lined arrow in Fig. 4 represents a concrete relationship in $t_{\text{M.in}}$. Each single-lined arrow in Fig. 4 represents an abstract relationship in $t_{\text{M.in}}$.

The component `camera` of type `Camera` and `intra_nw` of type `Switch` represent the network camera and the L2 switch, respectively. This part represents customer's environment. The two components `vs` of type `VideoSurveillance` and `hc` of type `HealthChecker` represent applications newly deployed. The functional requirements of these services are represented as the two abstract relationships of types `sendVideo` and `checkStatus`. The relationship `sendVideo` means that `camera` sends recorded video data to `vs`. The relationship `checkStatus` means that `hc` performs a regular health check of `camera`.

In addition to the functional requirement $t_{\text{M.in}}$, we assume the following quantitative requirement $QR$.

$QR\langle 1 \rangle$ The budget is within 2000 dollars.

$QR\langle 2 \rangle$ `vs` and `hc` require 7 and 2 gigabytes of RAM to provide stable operation, respectively.

$QR\langle 3 \rangle$ `Server_A` and `Server_B` have 6 and 14 gigabytes of RAM, respectively.

$QR\langle 4 \rangle$ The price of `Server_A` and `Server_B` is 800 and 1200 dollars, respectively.

Note that requirement $QR\langle 1 \rangle$ is imposed by the customer while requirements $QR\langle 2 \rangle$, $QR\langle 3 \rangle$ and $QR\langle 4 \rangle$ are derived from the nature of the software (i.e., `vs` and `hc`) and the hardware (i.e., `Server_A` and `Server_B`).

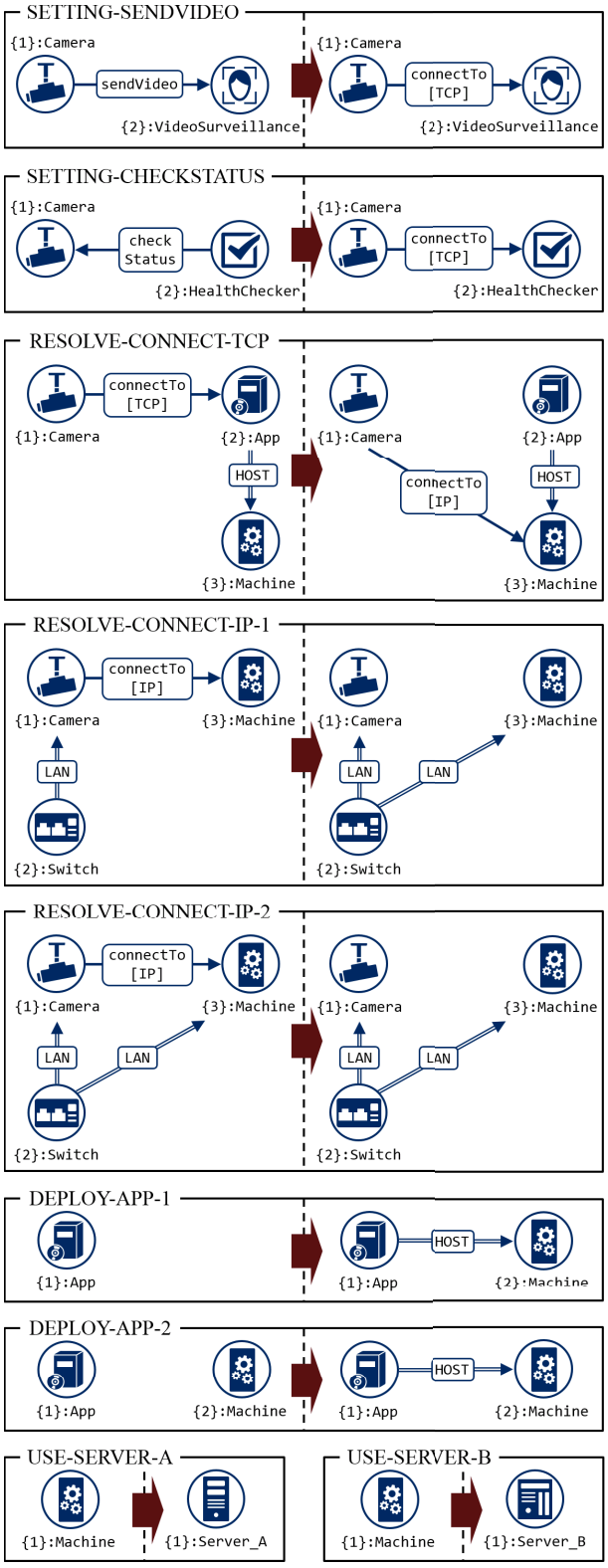


**Fig. 5** A set of refinement rules to refine our motivating example.

Figure 5 shows a list of refinement rules[†] used in the example. Rules **SETTING-SENDVIDEO** and **SETTING-CHECKSTATUS** resolve `sendVideo` and `checkStatus`, respectively, set values to the properties of the application, and impose `connectTo[TCP]`. Rule **RESOLVE-CONNECT-TCP** resolves `connectTo[TCP]` and imposes `connectTo[IP]`. Rule **RESOLVE-CONNECT-IP-1** also resolves `connectTo[IP]` and connects a `Machine` component and a `Switch` component with a concrete relationship `LAN`. Rule **RESOLVE-CONNECT-IP-2** resolves `connectTo[IP]` by using an existing connection between a `Machine` component and a `Switch` component. Rule **DEPLOY-APP-1** adds a new `Machine` component and connects an `App` component and the `Machine` component with a `HOST` relationship. Rule **DEPLOY-APP-2** connects an `App` component and a `Machine` component with a `HOST` relationship. Rules **USE-SERVER-A** and **USE-SERVER-B** concretize the type of a `Machine` component to `Server_A` and `Server_B`, respectively.

Figure 6 shows only a part of a search tree showing the search procedure of the basic search-based design method for service requirement $t_{M.in}$, extracting only paths leading to completely concrete topologies. The entire search tree branches into all states generated by applicable refinements and has much more intermediate states than the one shown in Fig. 6. As shown in Fig. 6, we can obtain six completely concrete topologies $t_{\mathrm{M.out.1}} \ldots t_{\mathrm{M.out.6}}$ by using the basic search-based design method described in Sect. 2. However, $t_{\mathrm{M.out.1}}$, $t_{\mathrm{M.out.3}}$, $t_{\mathrm{M.out.4}}$, and $t_{\mathrm{M.out.6}}$ violate the quantitative requirements.

In existing research on graph rewriting, a mechanism for including quantitative conditions in refinement rules has been proposed, and some graph rewriting tools (e.g., LMNtal [19], GROOVE [16]) have actually introduced such a mechanism. However, our motivating example does not correctly detect quantitative requirement violations by this method, because refinement rules can only describe quantitative conditions of a local part of graph.

For example, according to the aforementioned mechanism, the rule **DEPLOY-APP-1** would describe constraint $QR\langle 2\rangle$ and the rule **USE-SERVER-A** would describe constraint $QR\langle 3\rangle$ and $QR\langle 4\rangle$, respectively, and $QR\langle 2\rangle$ and $QR\langle 3\rangle$ would be individually judged when each rule is applied. However, in order to detect quantitative constraint violations in the topology $t_{\mathrm{M.out.1}}$, $QR\langle 2\rangle$ and $QR\langle 3\rangle$ must be referenced simultaneously and a decision must be made that "application `sv` cannot be deployed to machine `machine`<1>". For this reason, an existing graph rewriting framework for quantitative constraints cannot output a system design that satisfies all the quantitative requirements in the motivating example case.

Therefore, to correctly detect a violation of the quantitative constraints, we need a mechanism that can simultaneously check all the quantitative constraints imposed on the topologies. We implement such a mechanism in our proposed method, as discussed in the next section.

[†]For simplicity, we only show the minimum necessary refinement rules to be used in the example.

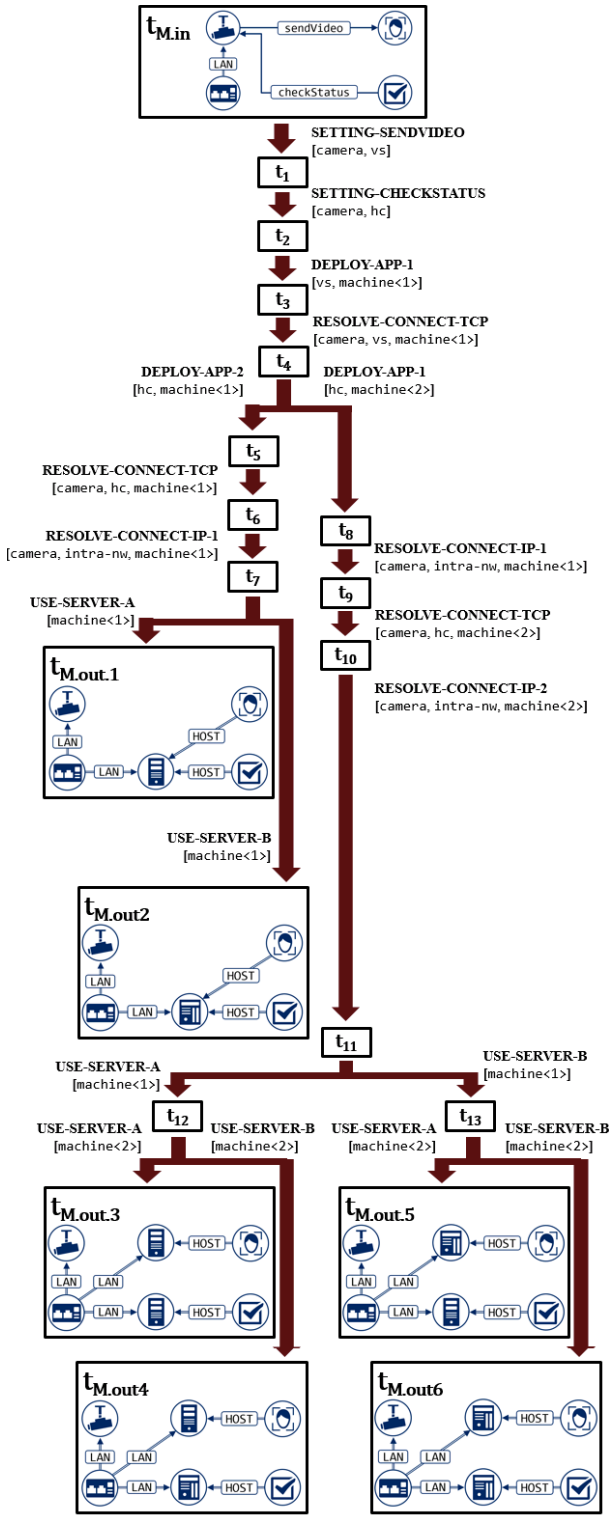


**Fig. 6** Six completely concrete topologies obtained by refining $t_{\text{M.in}}$. ($t_1 \ldots t_{13}$ are intermediate topologies occurring in the search process.)

## 4. Proposed Method

In this section, we extend the existing Search-based system design and propose a new method that can discover system configurations that meet service requirements, including both functional and quantitative requirements. The features of our method is the following three points.

- By associating *constraints on numeric values* to topologies, our method can include constraints that cannot be expressed in the graph format in the search state.
- By introducing set variables in addition to numeric variables, we were able to express the concept of accumulated value, such as the amount of memory used, the amount of network traffic and so on.
- By performing satisfiability check of quantitative constraints and pruning search states that violates quantitative constraints, our search algorithm can discover system configurations satisfying both functional and quantitative constraints effectively.

Here, we present the details of our design method. First, we explain how to integrate quantitative requirements with topologies and refinement rules. Second, we describe our new search-based design method. Finally, we explain a method to check the satisfiability of the quantitative requirements.

### 4.1 Constrained Topology and Refinement

In this section, we formalize constrained topologies and refinements. First, we need to formalize *quantitative constraints*. We define quantitative constraints as a set of *formulae* over numerical constants and two kind of variables: *set variables* and *numerical variables*. Note that set variables and numerical variables are disjoint, the domain of numerical variables is *non-negative numbers*, and the domain of set variables is *sets of numerical variables*, not sets of numbers.

**Definition 1.** *A* quantitative constraint *c is a set of* ⟨formula⟩ *defined as follows.*

$$
\begin{aligned}
\langle \text{formula} \rangle &::= \langle \text{set formula} \rangle \mid \langle \text{num formula} \rangle \\
\langle \text{num formula} \rangle &::= \langle \text{term} \rangle = \langle \text{term} \rangle \\
&\quad \mid \Sigma \langle \text{set variable} \rangle \leq \langle \text{term} \rangle \\
&\quad \mid \langle \text{term} \rangle \geq \langle \text{term} \rangle \mid \langle \text{term} \rangle \leq \langle \text{term} \rangle \\
\langle \text{set formula} \rangle &::= \langle \text{num variable} \rangle \in \langle \text{set variable} \rangle \\
&\quad \mid \langle \text{set variable} \rangle \subseteq \langle \text{set variable} \rangle \\
\langle \text{term} \rangle &::= \langle \text{constant} \rangle \mid \langle \text{num variable} \rangle \\
&\quad \mid \langle \text{term} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle \times \langle \text{term} \rangle
\end{aligned}
$$

*The symbols* $+, \times, \leq, \geq, \in$ *and* $\subseteq$ *mean "summation", "multiplication", "less than or equal", "greater than or equal", "is in", and "is subset of", respectively.*

*A set of all* ⟨set variable⟩ *in c are denoted by* $S_c$ *and a set of all* ⟨num variable⟩ *in c are denoted by* $N_c$.

*A set of all* ⟨set formula⟩ *in c are called the set-part of c and a set of all* ⟨num formula⟩ *in c are called the num-part*

*of c.*

We define the satisfiability of the quantitative constraints. A quantitative constraint $c$ is *satisfiable* when the following checking procedure succeeds.

1. Calculate the minimum solution of the set-part of $c$ (i.e., the minimum assignment of sets of numerical variables to set variables $\mu_c : S_c \to \{V \mid V \subseteq N_c\}$) such that all ⟨set formula⟩ are fulfilled under the assignment.
2. Replace all "$\Sigma S \leq$ ⟨term⟩" with "$v_1 + \cdots + v_n \leq$ ⟨term⟩", where $\mu_c(S) = \{v_1, \ldots, v_n\}$.
3. Check the existence of a solution of the num-part of $c$ (i.e., an assignment of non-negative numbers to numerical variables $M_c : N_c \to \mathbb{R}_{\geq 0}$) such that all ⟨formula⟩ in $c$ are fulfilled under the assignment.

When a quantitative constraint $c$ is satisfiable by $\mu_c$ and $M_c$, we call $(\mu_c, M_c)$ a solution of $c$.

**Example 2.** *Let's look at an example of a quantitative constraint, where* x, y *and* z *represent a num variable and* P, Q *and* R *represent a set variable.*

$$c_s = \{\mathtt{x} \geq 3, \mathtt{y} + 1 \geq \mathtt{x}, \mathtt{z} \leq \mathtt{y}, \Sigma\mathtt{R} \leq \mathtt{y}, \\ \mathtt{x} \in \mathtt{P}, \mathtt{z} \in \mathtt{Q}, \mathtt{P} \subseteq \mathtt{R}, \mathtt{Q} \subseteq \mathtt{R}\}$$

*The above quantitative constraint $c_s$ consists of eight formulae and is satisfiable by a solution* ({P ↦ {x}, Q ↦ {y}, R ↦ {x, y}}, {x ↦ 3, y ↦ 4, z ↦ 1}).

$$c_f = \{\mathtt{x} \geq 3, \mathtt{y} + 1 \geq \mathtt{x}, \mathtt{z} \geq \mathtt{y}, \Sigma\mathtt{R} \leq \mathtt{y}, \\ \mathtt{x} \in \mathtt{P}, \mathtt{z} \in \mathtt{Q}, \mathtt{P} \subseteq \mathtt{R}, \mathtt{Q} \subseteq \mathtt{R}\}$$

*The above quantitative constraint $c_f$ is not satisfiable. This is because the formulae* $\Sigma\mathtt{R} \leq \mathtt{y}$ *are replaced with* $\mathtt{x} + \mathtt{z} \leq \mathtt{y}$ *and* $\mathtt{x} + \mathtt{z} \leq \mathtt{y}$*, and* $\mathtt{z} \geq \mathtt{y}$ *are contradictory.*

We attach quantitative constraints to a topology to represent constraints on the quantitative parameters of components. A topology $t$ attached with a quantitative constraint $c$ is called a *constrained topology* and denoted $(t, c)$. We can represent quantitative requirements on topologies by attaching quantitative constraints, as in the following example.

**Example 3.** *As stated in Sect. 3, the topology representing the functional requirement of our motivating example is $t_{\text{M.in}}$. Now we also represent the quantitative requirement of our motivating example by attaching the following quantitative constraints $c_{\text{M.in}}$:*

$$c_{\text{M.in}} = \{\Sigma\mathtt{Budget} \leq 2000, \\ \mathtt{vs.req\_RAM} = 7, \mathtt{hc.req\_RAM} = 2\}$$

*where* Budget *is a set variable to contain the prices of all newly installed components, and* vs.req_RAM *and* hc.req_RAM *are numerical variables to represent the required RAM of* vs *and* hc*, respectively.*

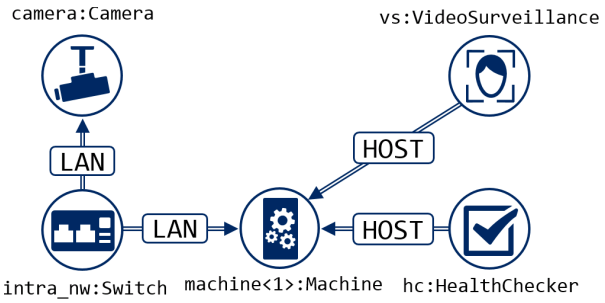


**Fig. 7** The topology $t_7$ shown in Fig. 6.

*Let us provide another example. Figure 7 is the topology $t_7$ occurred in Fig. 6. To represent a quantitative requirement on topology $t_7$, the following quantitative constraint $c_7$ is attached to $t_7$.*

$$c_7 = \{\Sigma\mathtt{Budget} \leq 2000, \\ \mathtt{vs.req\_RAM} = 7, \mathtt{hc.req\_RAM} = 2, \\ \mathtt{machine{<}1{>}.price} \in \mathtt{Budget}, \\ \mathtt{vs.req\_RAM} \in \mathtt{machine{<}1{>}.Used\_RAM}, \\ \mathtt{hc.req\_RAM} \in \mathtt{machine{<}1{>}.Used\_RAM}, \\ \Sigma\mathtt{machine{<}1{>}.Used\_RAM} \\ \leq \mathtt{machine{<}1{>}.limit\_RAM}, \\ \mathtt{machine{<}1{>}.price} \in \mathtt{Budget}\}$$

*The quantitative constraint $c_7$ includes $c_{\text{M.in}}$ and four additional formulae.* machine<1>.Used_RAM *is a set variable to contain all RAM used by applications installed on* machine<1>*,* machine<1>.limit_RAM *is a numerical variable to represent the limitation of available RAM, and* machine<1>.price *is a numerical variable to represent the price of* machine<1>*. These additional formulae mean that* vs *and* hc *require the RAM of* machine<1>*, the summation of* machine<1>.Used_RAM *is restricted to* machine<1>.limit_RAM*, and a certain budget is needed to install* machine<1>*.*

We also attach quantitative constraints to a refinement rule. A refinement rule $r$ attached to a quantitative constraint $c$ is called a *constrained refinement rule* and denoted $(r, c)$. A quantitative constraint attached to a refinement rule of size $n$ can use a variable that includes placeholders {1} ... {n} as a substring. We can replace placeholders {1} ... {n} in a quantitative constraint $c$ by using a matching $m = v_1, \ldots, v_n$. We denote the replaced quantitative constraint by $c[m]$.

As discussed later, a constrained refinement rule imposes the attached quantitative constraint to the target of refinement.

**Example 4.** *The following two quantitative constraints $c_{\text{USE-A}}$ and $c_{\text{USE-B}}$ are attached to rule* ***USE-SERVER-A*** *and* ***USE-SERVER-B****, respectively.*

$$c_{\text{USE-A}} = \{\{1\}.\mathtt{price} = 800, \{1\}.\mathtt{limit\_RAM} = 6\}$$
$$c_{\text{USE-B}} = \{\{1\}.\mathtt{price} = 1200, \{1\}.\mathtt{limit\_RAM} = 14\}$$

*Constraints* $c_{\text{USE-A}}$ *and* $c_{\text{USE-B}}$ *determines* {1}*'s price and RAM limitation as those of* Server_A *and* Server_B*, respectively.*

*Let us illustrate additional examples. The following two quantitative constraints* $c_{\text{DEPLOY-APP-1}}$ *and* $c_{\text{DEPLOY-APP-2}}$ *are attached to rules* ***DEPLOY-APP-1*** *and* ***DEPLOY-APP-2****, respectively.*

$$\begin{aligned} c_{\text{DEPLOY-APP-1}} = \{&\{2\}.\texttt{price} \in \texttt{Budget},\\ &\{1\}.\texttt{req\_RAM} \in \{2\}.\texttt{Used\_RAM},\\ &\Sigma\{2\}.\texttt{Used\_RAM} \leq \{2\}.\texttt{limit\_RAM}\}\\ c_{\text{DEPLOY-APP-2}} = \{&\{1\}.\texttt{req\_RAM} \in \{2\}.\texttt{Used\_RAM}\} \end{aligned}$$

*The quantitative constraints* $c_{\text{DEPLOY-APP-1}}$ *and* $c_{\text{DEPLOY-APP-2}}$ *represent a quantitative requirement for the right-hand side of* ***DEPLOY-APP-1*** *and* ***DEPLOY-APP-2****, respectively. The quantitative constraint* $c_{\text{DEPLOY-APP-2}}$ *is only one formula,* {1}.req_RAM ∈ {2}.Used_RAM*, which means an* App *component* {1} *requires the RAM of a* Machine *component* {2}*. In addition, the quantitative constraint* $c_{\text{DEPLOY-APP-1}}$ *has two more formulae:* {2}.price ∈ Budget *means that the installation cost of a new* Machine *component* {2} *comes out of the customer's budget* Budget*, and* Σ{2}.Used_RAM ≤ {2}.limit_RAM *means that the available RAM of* {2} *is restricted to* {2}.limit_RAM*.*

We call a person who defines refinement rules *a rule modeler*. The quantitative constraints for each refinement rule are defined by a rule modeler along with its graph transformation rule, and associated with the rule. For example, consider the case where a rule modeler defines **USE-SERVER-A** as a rule to convert the type of node {1} from abstract type Machine to concrete type Server_A. In this case, the rule modeler associates the quantitative constraint $c_{\text{USE-A}}$ in Example 4 with **USE-SERVER-A** as a constraint that sets undefined parameters (cf. RAM limit, etc...) of {1} to concrete Server_A's values.

**Example 5.** *In Sect. 3, we only discussed the quantitative requirements for memory usage and monetary cost, but other quantitative requirements can be also handled in our motivating example as follows.*

***CPU clock rate.*** *For example, a constraint that the operation of application* hc *requires CPU clock rate of 3.2 GH or higher can be expressed by adding constraint* hc.req_clockRate = 3.2 *to the quantitative requirement* $c_{\text{M.in}}$ *and associating additional constraint* {1}.req_clockRate ≤ {2}.clockRate *to refinement rule* ***DEPLOY-APP-1*** *and* ***DEPLOY-APP-2****. In addition, by associating additional constraint* {1}.clockRate = 2.0 *to refinement rule* ***USE-SERVER-A*** *and constraint* {1}.clockRate = 3.6 *to refinement rule* ***USE-SERVER-B****, we can express quantitative condition "application hc cannot work correctly on* Server_A*, while it can work correctly on* Server_B*."*

***Network traffic.*** *Network traffic can be handled in the same way as the amount of memory used. For example, consider the case where camera* C *uses 1 Mbps of traffic when sending video to application* vs*. In this case, we can express network traffic as constraints by associating the following constraint* $c_{\text{TCP}}$ *with refinement rule* ***RESOLVE-CONNECT-TCP****.*

$$\begin{aligned} c_{\text{TCP}} = \{&\{1\}.\texttt{tf} \in (\{1\},\{2\}).\texttt{all\_TCP\_tf}\\ &\sum (\{1\},\{2\}).\texttt{all\_TCP\_tf}\\ &\leq (\{1\},\{2\}).\texttt{TCP\_tf}\} \end{aligned}$$

*Here, variable* {1}.tf *means the amount of traffic required for sending the video data of camera* {1}*, variable* ({1},{2}).all_TCP_tf *is a set variable including num variables of all TCP communications between camera* {1} *and application* {2}*, and variable* ({1},{2}).TCP_tf *means the amount of traffic required to make all TCP communications between camera* {1} *and application* {2}*.*

*Constraint* $c_{\text{TCP}}$ *represents the amount of network traffic of TCP communication, but similarly, constraints on the amount of network traffic of other network layers can also be represented.*

Now we explain the procedure for refining constrained topologies. Let $(t, c_t)$ be a constrained topology, $(r, c_r)$ be a constrained refinement rule, and $m$ be a matching such that the action $r[m]$ is applicable to $t$. The procedure for refining $(t, c_t)$ by $(r, c_r)$ and $m$ is defined as follows:

1. Obtain $r[m](t)$ by the plain refinement procedure described in Sect. 2.
2. Check the satisfiability of $c_t \cup c_r[m]$. (If $c_t \cup c_r[m]$ is not satisfiable, this refinement procedure fails.)
3. Attach $c_t \cup c_r[m]$ to $r[m](t)$.

We obtain a constrained topology $(r[m](t), c_t \cup c_r[m])$ and denote it by $(r, c_r)[m]((t, c_t))$.

**Example 6.** *Let us consider the refinement of the constrained topology* $(t_7, c_7)$ *by a constrained refinement rule* (**USE-SERVER-B**, $c_{\text{USE-B}}$) *and a matching* $m$ = machine<1>*. First, as shown in Fig. 6, refinement by* ***USE-SERVER-B*** *and* $m$ *converts* $t_7$ *into* $t_{\text{M.out.2}}$*. Second,* $c_7$ *is merged with the following quantitative constraint* $c_{\text{USE-B}}[m]$*.*

$$\begin{aligned} c_{\text{USE-B}}[m] = \{&\texttt{machine<1>.price} = 1200,\\ &\texttt{machine<1>.limit\_RAM} = 14\} \end{aligned}$$

*As a result, the following constraint* $c_{\text{M.out.2}}$ *is obtained.*

$$\begin{aligned} c_{\text{M.out.2}} = \{&\Sigma\texttt{Budget} \leq 2000,\\ &\texttt{vs.req\_RAM} = 7, \texttt{hc.req\_RAM} = 2,\\ &\texttt{machine<1>.price} \in \texttt{Budget},\\ &\texttt{vs.req\_RAM} \in \texttt{machine<1>.Used\_RAM},\\ &\texttt{hc.req\_RAM} \in \texttt{machine<1>.Used\_RAM},\\ &\Sigma\texttt{machine<1>.Used\_RAM}\\ &\qquad \leq \texttt{machine<1>.limit\_RAM} \end{aligned}$$

$$\begin{aligned}&\texttt{machine<1>.price} \in \texttt{Budget},\\ &\texttt{machine<1>.price} = 1200,\\ &\texttt{machine<1>.limit\_RAM} = 14\}\end{aligned}$$

*Finally, the satisfiability of $c_{\text{M.out.2}}$ is checked. The following assignment $(\mu_{\text{M.out.2}}, M_{\text{M.out.2}})$ satisfies $c_{\text{M.out.2}}$, so $c_{\text{M.out.2}}$ is proven to be satisfiable and this refinement procedure succeeds.*

$$\begin{aligned}\mu_{\text{M.out.2}} = \{&\texttt{machine<1>.Used\_RAM} \mapsto\\ &\{\texttt{vs.req\_RAM}, \texttt{hc.req\_RAM}\}\}\\ M_{\text{M.out.2}} = \{&\texttt{vs.req\_RAM} \mapsto 7, \texttt{hc.req\_RAM} \mapsto 2,\\ &\texttt{machine<1>.price} \mapsto 1200,\\ &\texttt{machine<1>.limit\_RAM} \mapsto 14\}\end{aligned}$$

*Next, we introduce a failure case of the refinement procedure. Consider the refinement of the constrained topology $(t_7, c_7)$ by a constrained refinement rule* (**USE-SERVER-A**, $c_{\text{USE-A}}$) *and a matching $m$ =* `machine<1>`. *In this case, in the second step of the refinement procedure, the following constraint $c_{\text{M.out.1}}$ is obtained.*

$$\begin{aligned}c_{\text{M.out.1}} = \{&\Sigma\texttt{Budget} \le 2000,\\ &\texttt{vs.req\_RAM} = 7, \texttt{hc.req\_RAM} = 2,\\ &\texttt{machine<1>.price} \in \texttt{Budget},\\ &\texttt{vs.req\_RAM} \in \texttt{machine<1>.Used\_RAM},\\ &\texttt{hc.req\_RAM} \in \texttt{machine<1>.Used\_RAM},\\ &\Sigma\texttt{machine<1>.Used\_RAM}\\ &\qquad \le \texttt{machine<1>.limit\_RAM}\\ &\texttt{machine<1>.price} \in \texttt{Budget},\\ &\texttt{machine<1>.price} = 800,\\ &\texttt{machine<1>.limit\_RAM} = 6\}\end{aligned}$$

*The quantitative constraint $c_{\text{M.out.1}}$ is not satisfiable, because the formula*

$$\begin{aligned}&\Sigma\texttt{machine<1>.Used\_RAM}\\ &\quad \le \texttt{machine<1>.limit\_RAM}\end{aligned}$$

*is replaced by*

$$\begin{aligned}&\texttt{vs.req\_RAM} + \texttt{hc.req\_RAM}\\ &\quad \le \texttt{machine<1>.limit\_RAM}\end{aligned}$$

*and is in conflict with the following equalities.*

$$\begin{aligned}&\texttt{vs.req\_RAM} = 7, \texttt{hc.req\_RAM} = 2,\\ &\texttt{machine<1>.limit\_RAM} = 6\end{aligned}$$

*Thus, this refinement procedure is failed.*

*This failure means that the refinement of $t_7$ by **USE-SERVER-A** and the matching $m$ =* `machine<1>` *shown in Fig. 6 does not occur. Therefore, our method does not output the topology $t_{\text{M.out.1}}$ that violates the RAM requirements of the applications.*

### 4.2 Search Algorithm on Constrained Topologies

In this section, we propose a new search-based design algorithm that performs a tree search on constrained topologies by means of constrained refinement rules in a similar way to the method described in Sect. 2. The key difference here is that our algorithm refines a quantitative constraint in addition to a topology and excludes candidates from the search if the quantitative constraint is not satisfiable.

Algorithm 1 shows our search-based design algorithm. It receives a constrained topology representing the service requirement and repeatedly applies the new refinement procedure to the constrained topology. The function $\sigma$ in line 3 is called a tree-search strategy receiving search candidates $T$ and returning a subset of $\{((t, c_t), (r, c_r), m) \mid (t, c_t) \in T, r[m] \text{ is applicable to } t\}$. This function chooses targets of refinement in each iteration. By line 9, a quantitative constraint is updated, and by line 10, a constrained topology can be added to search candidates only when the quantitative constraint is proven to be satisfiable. In our algorithm, all quantitative constraints given as customer requirements and added by quantitative refinement rules are propagated to child nodes of the search tree and are checked for consistency when new search candidates are added to the tree. Therefore, by using this algorithm, we can reliably obtain only completely concretized topologies that satisfy both the functional and quantitative requirements.

Here, we explain why our algorithm performs constraint checking on all intermediate states. In fact, by replacing conditional expressions of the if statements located in lines 10 and 11, constraints is only checked for completely concrete topologies and the number of constraint checks is reduced. However, if a search algorithm ignores the satisfiability of quantitative requirements during search process, it may result in a large number of topologies that satisfy functional requirements but violate quantitative requirements. For example, the problem used in Sect. 5 with $n = 7$ has 342 concrete configurations that satisfy the functional requirements, but only one of them satisfies the quantitative requirements. Since it takes a considerable amount of time to arrive at a concrete configuration, it is very inefficient to generate and test these concrete configurations one by one at random. Due to the above reason, we adopted a method that detects the violation of quantitative requirements in intermediate states and removes them from search candidates, as in the proposed method.

**Example 7.** *Figure 8 shows a part of the search process when constrained topology $(t_{\text{M.in}}, c_{\text{M.in}})$ is given as input of Algorithm 1. Each $t_i$ is the same topologies shown in Fig. 6 and each $c_i$ is the quantitative constraint attached to $t_i$.*

*Let us focus on constrained topology $(t_7, c_7)$. Topology $t_7$ is shown in Fig. 7 and constraint $c_7$ is shown in **Example 6**.*

*As shown in Fig. 6, **USE-SERVER-A**[`machine<1>`] is applicable to $t_7$ and $t_{\text{M.out.1}}$ is generated as a re-*

*sult of refinement of $t_7$ and violates quantitative requirement QR⟨2⟩. In contrast, as we stated in **Example** 6, constraint $c_7 \cup c_{USE\text{-}A}$[machine<1>] is not satisfiable and the refinement procedure of $(t_7, c_7)$ by $(\textbf{USE-SERVER-A}, c_{USE\text{-}A})$[machine<1>] is failed.*

*On the other hand, as we stated in **Example** 6, constraint $c_7 \cup c_{USE\text{-}B}$[machine<1>] is satisfiable. So the constrained topology $(t_{\text{M.out.2}}, c_{\text{M.out.2}})$ is generated as a result of refinement of $(t_7, c_7)$.*

*Similarly, $t_{10}$ and $t_{\text{M.out.6}}$ in Fig. 6 do not occur in Fig. 8 because $c_{11} \cup c_{USE\text{-}A}$[machine<1>] and $c_{13} \cup c_{USE\text{-}B}$[machine<2>] is not satisfiable, respectively. As a result, Algorithm 1 can output only $t_{\text{M.out.2}}$ and $t_{\text{M.out.5}}$ from $t_{\text{M.out.1}}, \ldots, t_{\text{M.out.6}}$ as system configurations that satisfy all the requirements in the motivating example.*

**Algorithm 1** Tree Search on Constrained Topologies

**Input:** Requirement $(t_0, c_0)$
**Output:** Service configuration $t_{sc}$, or **"Failed"**

```
1:  T ← {(t0, c0)}
2:  loop
3:      E ← σ(T)
4:      if E = ∅ then
5:          return "Failed"
6:      end if
7:      for ((tl, cl), (tr, cr), m) ∈ E do
8:          t' ← r[m](t)
9:          c' ← cl ∪ cr[m]
10:         if c' is satisfiable then
11:             if t' is completely concrete then
12:                 return t'
13:             end if
14:             T ← T ∪ {(t', c')}
15:         end if
16:     end for
17: end loop
```

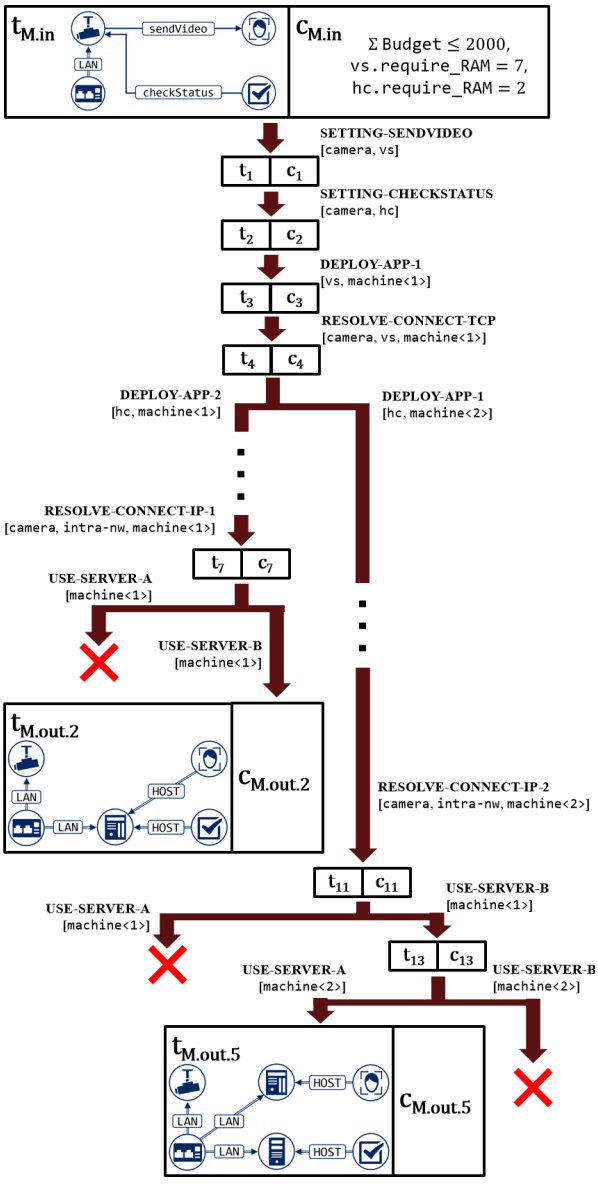


**Fig. 8** A part of search process of Algorithm 1.

## 4.3 Efficient Algorithm of Constraint Checking

In this section, we present *constraint checking*, which is a technique to judge whether a given quantitative constraint is satisfiable at line 10 in Algorithm 1. To check the satisfiability of a quantitative constraint $c$ by the procedure described in Sect. 4.1, we need a way to obtain $\mu_c$, which is the minimum assignment of the set-part of $c$, and a way to check the satisfiability of the num-part of $c$. Our definition of quantitative constraints is a very general one, so there are several available methods for resolving them.

First, we calculate the set-part of a given quantitative constraint. Assignment of values to set variables that satisfy given set constraints is easily calculated by the round-robin iterative algorithm [11].

Second, a problem to check the satisfiability of numerical constraints is called a *non-linear real arithmetic (NRA) problem*, and there are many algorithms [7] and tools [6], [8], [9] to resolve it. One of these is Z3 [8], a state-of-the-art tool developed by Microsoft. We use Z3 in our prototype to check the satisfiability of the num-part of a quantitative constraint.

By using the round-robin algorithm and an external tool for solving NRA, we obtain a *naïve* algorithm $C_{\text{naïve}}$. However, it is generally very time-consuming to solve NRA problems, and thus it is inefficient to run an NRA solving process in each loop iteration of Algorithm 1.

To address this issue, we propose an efficient constraint checking algorithm $C_{\text{opt}}$ that omits a part of the satisfiability checking for numerical constraints. Algorithm 2 shows the procedure of $C_{\text{opt}}$, which is based on the following fact pertaining to our quantitative constraints.

**Fact 1.** *Let $c_1$ and $c_2$ be quantitative constraints. If $c_1$ is a subset of $c_2$ and $c_2$ is satisfiable, $c_1$ is satisfiable.*

Fact 1 enables us to omit a part of the satisfiability checking. That is, when a quantitative constraint $c_s$ is

proven to be satisfiable, we can judge that all subsets of $c_s$ are satisfiable as well, without performing additional constraint checking. In addition, when a quantitative constraint $c_f$ is proven to be not satisfiable, we can judge that all supersets of $c_f$ are also not satisfiable. By using Fact 1, algorithm $C_{\text{opt}}$ memoizes the results of constraint checking to **SATs** and **UNSATs** and can omit a part of the constraint checking by using these memoized checking results.

**Algorithm 2** Efficient numerical constraint checking $C_{opt}$

**Input:** A num-part of a quantitative constraint $c_{\text{num}}$ and global variables **SATs** and **UNSATs**
**Output:** Satisfiability of $c_{\text{num}}$: **true** or **false**

```
for all c_s ∈SATs do
    if c_num ⊆ c_s then
        return true
    end if
end for
for all c_f ∈UNSATs do
    if c_f ⊆ c_num then
        return false
    end if
end for
result ← output of an external checking tool for c_num
if result then
    SATs ← SATs ∪{c_num}
else
    UNSATs ← UNSATs ∪{c_num}
end if
```

## 5. Evaluation

We implemented our search-based design algorithm in Python 3 and executed it on a server with Intel-Xeon (3.60 GHz) and 32 GB of memory. We prepared two prototype tools, $\mathcal{T}_{\text{opt}}$ and $\mathcal{T}_{\text{naïve}}$, for implementing our method. Tool $\mathcal{T}_{opt}$ implements checking algorithm $C_{\text{opt}}$ described in Sect. 4, and tool $\mathcal{T}_{\text{naïve}}$ implements checking algorithm $C_{\text{naïve}}$. We tested the effectiveness of algorithm $C_{\text{opt}}$ by comparing the evaluation results of $\mathcal{T}_{\text{opt}}$ and $\mathcal{T}_{\text{naïve}}$. We adopted Z3 [8] as a solver to check satisfiability of the num-part of a quantitative constraint.

We conducted the experiment under scenarios based on our motivating example described in Sect. 3 and added difficulty by having the tools find the appropriate system designs. We applied our prototype tool to experimental inputs and evaluated whether the desired system configuration could be obtained. We also examined the time efficiency.

### 5.1 Evaluation Setup

#### 5.1.1 Service Requirement

The experiment used the same components as the motivating example described in Sect. 3, and also used the same quantitative requirements $QR\langle 2\rangle$, $QR\langle 3\rangle$ and $QR\langle 3\rangle$; ($QR\langle 2\rangle$) nodes `VideoSurveillance` and `HealthChecker` require 7 and 2 gigabytes of RAM, respectively, ($QR\langle 3\rangle$) `Server_A` and `Server_B` have 6 and 14 gigabytes of RAM, respectively, and ($QR\langle 4\rangle$) the price of `Server_A` and `Server_B` is 800 and 1200 dollars, respectively.

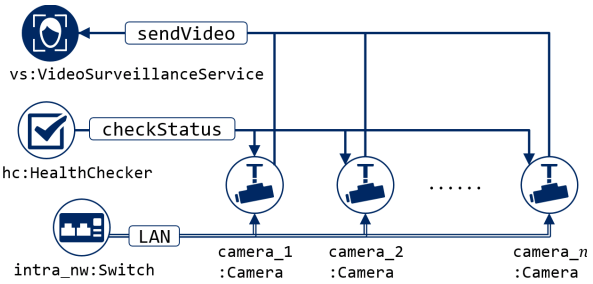


**Fig. 9** A topology $t_{\text{Eval}.n}$ used in evaluation.

Figure 9 shows a topology $t_{\text{Eval}.n}$ used as a service requirement in the evaluation. While the motivating example $t_{M.in}$ has only one camera, topology $t_{\text{Eval}.n}$ has $n(\geq 1)$ cameras, and the number of video surveillance applications needs to be scaled up accordingly. The video surveillance application we used can handle more than one camera, but 3 gigabytes of RAM per camera is required in addition to 7 gigabytes of RAM for basic usage. The topology $t_{\text{Eval}.n}$ has a `vs` component of type `VideoSurveillanceService`, whereas the motivating example $t_{M.in}$ has a `vs` component of type `VideoSurveillance`. The `VideoSurveillanceService` type intuitively represents *a service function* composed of multiple components of type `VideoSurveillance`.

The following constraint $c_{\text{Eval}.n}$ is attached to $t_{\text{Eval}.n}$.

$$c_{\text{Eval}.n} = \{\texttt{hc.req\_RAM} = 2, \Sigma\texttt{Budget} \leq Cost_{\min}(n)\}$$

Here, $Cost_{\min}(n)$ is defined as

$$Cost_{\min}(n) := \begin{cases} 1200i & \text{if } n = 2i-1 \\ 1200i + 800 & \text{if } n = 2i. \end{cases}$$

The value of $Cost_{\min}(n)$ is the minimum budget required to build a system satisfying the service requirement $t_{\text{Eval}.n}$.

#### 5.1.2 Refinement Rules

The refinement rules used in the evaluation are basically the same as those introduced in Sect. 3, except for the two points described below.

First, the following quantitative constraint $c_{\text{SETTING-SENDVIDEO}}$ is attached to rule **SETTING-SENDVIDEO**. The constraint $c_{\text{SETTING-SENDVIDEO}}$ is intended to take into account the additional required RAM when a new camera is added to the target of the `VideoSurveillance` components.

$$c_{\text{SETTING-SENDVIDEO}} = \{\{1\}.\texttt{req\_RAM} = 3, \{1\}.\texttt{req\_RAM} \in \{2\}.\texttt{consumed\_RAM}\}$$

Second, we add the two rules **SCALE-UP-1** and **SCALE-UP-2**, which are shown in Fig. 10. By using either of these rules, the component `VideoSurveillanceService`

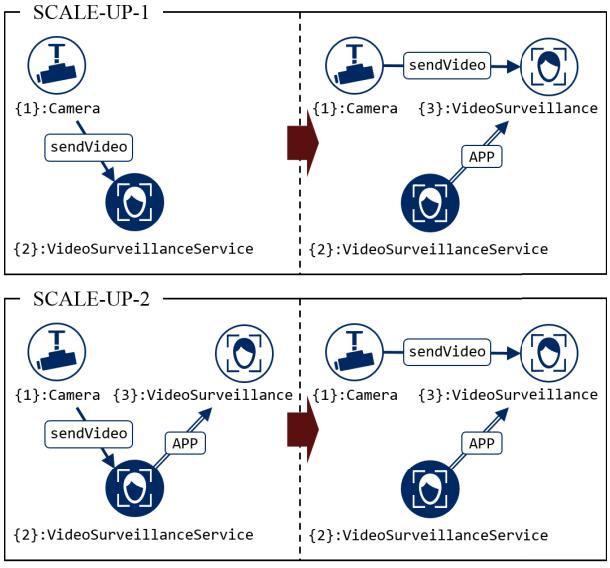


**Fig. 10** Two additional rules used in evaluation.

can generate a new component of type `VideoSurveillance` and assign a request from a camera (i.e., `sendVideo`) to components of type `VideoSurveillance`.

The following quantitative constraint $c_{\text{SCALE-UP-1}}$ is attached to rule **SCALE-UP-1**.

$$c_{\text{SCALE-UP-1}} = \{\{3\}.\texttt{base\_RAM} = 7,\ \Sigma\{3\}.\texttt{consumed\_RAM} \leq \{3\}.\texttt{req\_RAM},\ \{3\}.\texttt{base\_RAM} \in \{3\}.\texttt{consumed\_RAM}\}$$

The quantitative constraint $c_{\text{SCALE-UP-2}} = \emptyset$ is attached to rule **SCALE-UP-2**.

### 5.1.3 Expected System Configuration

Figure 11 shows an expected system configuration $t_{\text{Res}.n}$ obtained by refining $(t_{\text{Eval}.n}, c_{\text{Eval}.n})$.

When n is equal to $2i (i = 1, 2, \ldots)$, the "Type A" topology shown in Fig. 11 is the only system configuration that satisfies the requirement $(t_{\text{Eval}.n}, c_{\text{Eval}.n})$. Alternatively, when n is equal to $2i - 1 (i = 1, 2, \ldots)$, the "Type B" topology shown in Fig. 11 is the only system configuration that satisfies the requirement $(t_{\text{Eval}.n}, c_{\text{Eval}.n})$. We expect our prototype tools to output the topology $t_{\text{Res}.n}$ by applying our search-based design method to requirements $(t_{\text{Eval}.n}, c_{\text{Eval}.n})$.

## 5.2 Results

Our prototype tool could successfully generate the topology $t_{\text{Res}.n}$ from each $(t_{\text{Eval}.n}, c_{\text{Eval}.n})$ for $n = 1, \ldots, 7$. Tables 1 and 2 list the evaluation results. The "Total time [sec]" column shows the total time it took for the tools to find the topology $t_{\text{Res}.n}$. The "Time to check [sec]" column shows the summation of the time it took for the tools to check the satisfiability of the quantitative constraints at each refinement. The "# of check" column shows the total number of satisfiability checks performed while the tools searched the topology $t_{\text{Res}.n}$.

As shown, checking algorithm $C_{\text{opt}}$ was much more efficient than naïve checking algorithm $C_{\text{naïve}}$. Algorithm $C_{\text{opt}}$ could reduce the computation time of the checking by 73.3–89.9% and the number of checks by 75.4–94.6% compared to the conventional algorithm.

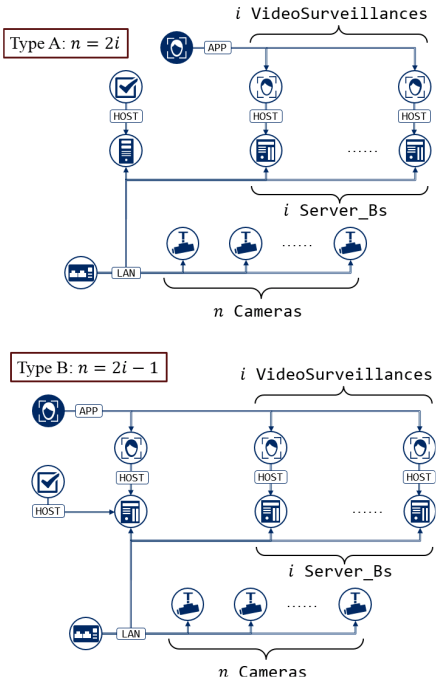


**Fig. 11** A topology $t_{\text{Res}.n}$ expected as an output of our algorithm.

**Table 1** Evaluation results of $\mathcal{T}_{\text{opt}}$.

| n | Total time [sec] | Time to check [sec] | # of check |
|---|---|---|---|
| 1 | 1.22 | 0.44 | 24 |
| 2 | 2.19 | 1.14 | 62 |
| 3 | 5.48 | 3.52 | 180 |
| 4 | 11.89 | 8.25 | 391 |
| 5 | 36.26 | 24.53 | 961 |
| 6 | 219.6 | 130.97 | 3453 |
| 7 | 2622.49 | 1706.42 | 21829 |

**Table 2** Evaluation results of $\mathcal{T}_{\text{naïve}}$.

| n | Total time [sec] | Time to check [sec] | # of check |
|---|---|---|---|
| 1 | 3.66 | 2.81 | 161 |
| 2 | 6.01 | 4.91 | 287 |
| 3 | 15.64 | 13.58 | 756 |
| 4 | 34.45 | 31 | 1590 |
| 5 | 152.49 | 140.49 | 6248 |
| 6 | 1389.12 | 1297.19 | 47868 |
| 7 | 12302.64 | 11345.12 | 407035 |

## 6. Conclusion

We have presented a search-based system design method that receives the functional and quantitative service requirements at the same time and generates a system configuration that satisfies them both. Our method propagates con-

straints over the quantitative parameters of system components along with a search process to find the system configuration and chooses only topologies with consistent quantitative constraints as search candidates. We also proposed an efficient constraint checking algorithm for checking the consistency of quantitative constraints during the search process and showed that it decreases the computation time for constraint checking.

Although the proposed method can deal with quantitative constraints, it still has many limitations and challenges. In particular, there are three major challenges. First, Preparing and maintaining constrained refinement rules often requires a high degree of expertise or a high human cost. For example, if we want to add additional constraints on different quantitative parameters to the motivating example, we have to add constraints to all the relevant refinement rule. Second, it is not clear what kinds of constraints should be handled in order to express practical quantitative requirements. We should experiment with handling a greater variety of quantitative requirements in more various scenarios. The third is the time cost of constraint checking: even though we have achieved a significant improvement in checking efficiency as shown in Sect. 5, the time spent on constraint checking still takes up a large part of the computation time. We plan to enhance constraint checking mechanism by using characteristics of quantitative requirements.

## Acknowledgments

This work was conducted as part of the project entitled "Research and development for innovative AI network integrated infrastructure technologies (JPMI00316)" supported by the Ministry of Internal Affairs and Communications, Japan.

### References

[1] Kustomize - kubernetes native configuration management. https://kustomize.io/
[2] Network orchestration & edge networking — cloudify. https://cloudify.co/
[3] Openstack docs: Heat orchestration template (HOT) guide. https://docs.openstack.org/heat/rocky/template_guide/hot_guide.html
[4] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C. Nechifor, D. Petcu, A. Gericke, and C. Sheridan, "MODAClouds: A model-driven approach for the design and execution of applications on multiple clouds," 2012 4th International Workshop on Modeling in Software Engineering (MISE), pages 50–56, June 2012.
[5] L. Baresi, R. Heckel, S. Thöne, and D. Varro, "Style-based modeling and refinement of service-oriented architectures," Softw. Syst. Model., vol.5, pp.187–207, 2006.
[6] C. Barrett, C.L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, "CVC4," G. Gopalakrishnan and S. Qadeer, eds., Computer Aided Verification, pp.171–177, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
[7] G.E. Collins, "Quantifier elimination for real closed fields by cylindrical algebraic decompostion," H. Brakhage, ed., Automata Theory and Formal Languages, pp.134–183, Springer Berlin Heidelberg, Berlin, Heidelberg, 1975.
[8] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol.4963, pp.337–340, April 2008.
[9] B. Dutertre, "Yices 2.2," A. Biere and Ro. Bloem, eds., Computer Aided Verification, pp.737–744, Springer International Publishing, Cham, 2014.
[10] Á. Hegedüs, Á. Horváth, I. Ráth, and D. Varró, "A model-driven framework for guided design space exploration," 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp.173–182, 2011.
[11] J.B. Kam and J.D. Ullman, "Global data flow analysis and iterative algorithms," J. ACM, vol.23, no.1, pp.158–171, 1976.
[12] T. Kuroda, T. Kuwahara, T. Maruyama, K. Satoda, H. Shimonishi, T. Osaki, and K. Matsuda, "Weaver: A novel configuration designer for IT/NW services in heterogeneous environments," 2019 IEEE Global Communications Conference (GLOBECOM), pp.1–6, 2019.
[13] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," IEEE Trans. Softw. Eng., vol.30, no.5, pp.311–327, 2004.
[14] A.M. Maia, Y. Ghamri-Doudane, D. Vieira, and M.F. de Castro, "Optimized placement of scalable IoT services in edge computing," 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp.189–197, 2019.
[15] D. Menascé, H. Gomaa, S. Malek, and J. Sousa, "SASSY: A framework for self-architecting service-oriented systems," IEEE Softw., vol.28, no.6, pp.78–85, Jan. 2012.
[16] A. Rensink, I. Boneva, H. Kastenberg, and T. Staijen, "User manual for the GROOVE tool set," https://groove.ewi.utwente.nl/wp-content/uploads/usermanual1.pdf, 2009.
[17] M. Rutkowski, L. Boutier, and C. Lauwers, "TOSCA simple profile in YAML version 1.2," https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/TOSCA-Simple-Profile-YAML-v1.2.html, 2019.
[18] G. Sallam and B. Ji, "Joint placement and allocation of virtual network functions with budget and capacity constraints," IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp.523–531, 2019.
[19] K. Ueda, "LMNtal as a hierarchical logic programming language," Theor. Comput. Sci., vol.410, no.46, pp.4784–4800, Nov. 2009.

**Takuya Kuwahara** received his master's degree of information science and technology from Graduate School of Information Science and Technology, The University of Tokyo in 2015 and has been engaged in research on formal methods for program verification. He joined in NEC Corporation in 2015. Now he is working on researches for automation technology for ICT system design and operation.

**Takayuki Kuroda** received M.E and Ph.D. degreed from the Graduate School of Information Science, Tohoku University, Sendai, Japan in 2006 and 2009. He joined NEC Corporation in 2009 and has been engaged in research on model-based system management for Cloud application and Software-defined networks. As a visiting scalar in the Electrical Engineering and Computer Science department at the Vanderbilt University in Nashville, he studied declarative approach of automated workflow generation for ICT system update. Now he is working on research for automation technologies for system design, optimization and operation.

**Takao Osaki** received Ph.D. from the Graduate School of Science, Osaka University, Osaka, Japan in 1999. He joined NEC Corporation in 1999 and has been engaged in research and development on requirement engineering and enterprise computer system integration.

**Kozo Satoda** received his B.E and M.E degrees in electrical engineering from Kyoto University in 1991 and 1993 respectively. He joined NEC in 1993. He has received best paper award of IEEE CQR workshop 2010, best paper award of IEEE CCNC 2017, IEICE Communications Society Excellent Paper Award 2016, 63rd Electrical Science and Engineering Promotion Awards and 2016 IPSJ Industrial Achievement Award. His research interests include multimedia communication, streaming and mobile traffic management. He is a member of IEICE, IPSJ and IEEE.