On the Complexity of Reconfiguration in Systems with Legacy Components

Jacopo Mauro, Gianluigi Zavattaro

On the Complexity of Reconfiguration in Systems with Legacy Components

Jacopo Mauro and Gianluigi Zavattaro

Department of Computer Science and Engineering - Univ. of Bologna / INRIA

Abstract. In previous works we have proved that component reconfiguration in the presence of conflicts among components is non-primitive recursive, while it becomes poly-time if there are no conflicts and under the assumption that there are no components in the initial configuration. The case with non-empty initial configurations was left as an open problem, that we close in this paper by showing that, if there are legacy components that cannot be generated from scratch, the problem turns out to be PSpace-complete.

1 Introduction

Modern software systems are obtained as combination of software artefacts having complex interdependencies. Their composition, configuration and management is a difficult task, traditionally performed manually or by writing low level configuration scripts. Recently, many high level languages and tools like, for instance, TOSCA [21], Juju [15] or Engage [9] have been proposed to support the application manager in this difficult task. By adopting these tools, it is possible to describe the software components required to realise the system, define their interdependencies and specify the configuration actions to be executed to actually deploy an instance of the desired system. In some limited cases (for instance in Engage and in the declarative modality of TOSCA), and under some specific assumptions (no circular component dependencies), such tools automatically synthesise the configuration actions to be executed. Automatic deployment is becoming more and more important for these tools especially due to the advent of virtualization technologies, like in Cloud Computing, that makes it possible to quickly acquire and release computing resources in order to deploy new software systems or reconfigure running applications on-demand.

In previous works [6,8,18] we have performed a rigorous and systematic analysis of the automatic deployment problem. We have proved that in general the problem is undecidable, it is non-primitive recursive if component interdependencies do not include numerical constraints, and it is poly-time if also conflicts among components are not considered. This last result was proved by restricting our attention on the deployment of an application from scratch, that is, by assuming that the initial configuration is empty. This result is of particular interest because it underpins the recent industrial trend of using the so called "immutable servers" [20]. The application is divided in stateless components/services that are deployed on virtual machines. When a new version of the component is developed or the virtual machine needs updates (e.g., new security patches have to be installed), instead of upgrading in-place the virtual machine, a new one is created and the old one destroyed. According to this approach, since all the needed components can be freshly generated, the result proven in [18] shows that a new deployment can be efficiently computed simply generating a new configuration from scratch, without considering or reusing existing components.

Unfortunately the "immutable servers" approach has also some disadvantages. First of all, it requires that every application is carefully designed to ensure that important data is stored and not lost when the old servers are destroyed. System upgrades are usually slower because creating new virtual servers takes more time than performing an upgrade in-place. But, most importantly, this approach cannot be adopted in presence of legacy components, a scenario that often happens in practice due to software applications that for several reasons, like incompatibility with novel computing architectures or cost purposes, cannot be replaced and must be kept in-place.

Given these premises, the following question arises. How complex is the *reconfiguration* problem of deciding if a final configuration can be reached in the presence of components that cannot be switched off and re-deployed from scratch? The goal of this paper is to address this last question, proving that *reconfiguration* is no longer polynomial, but it turns out to be PSpace-complete.

More precisely, we first report the formalisation of the *reconfiguration* problem using the Aeolus component model adopted in [18] (Sect. 2). Then we show that the problem can be solved by performing a symbolic forward search of the new configurations that can be reached from a given initial one (Sect. 3). The symbolic approach allows for a finite representation of all the (possibly infinite) reachable configurations. Unfortunately, the number of possible symbolic configurations is exponential; we mitigate this blow up by adopting a nondeterministic polynomial-space visit of the (symbolic) search space. Finally, we show that it is not possible to significantly improve our algorithm as we prove that the reconfiguration problem is indeed PSpace-hard (Sect. 4). The proof is by reduction from the reachability problem in 1-safe Petri nets [3].

Proofs are reported in separate Appendixes, for reviewer convenience.

2 Formalising the Reconfiguration Problem

In this section we recapitulate the fragment of the Aeolus model used to formally define the reconfiguration problem. This fragment of Aeolus [18] is exactly the one used by the planner Metis [17], a tool for finding deployment plans starting from an empty initial configuration integrated in an industrial deployment platform [7].[1] In the Aeolus model, a component is a grey-box showing relevant internal states and the actions that can be acted on the component to change its

[1] W.r.t. the Aeolus model [6], the fragment used by Metis does not allow the use of capacity constraints, conflicts, and multiple state changes.

state during (re)configuration. Each state activates provide-ports and require-ports representing functionalities that the component provides and needs. Active require-ports must be bound to active provide-ports of other components.

The problem that we address in this paper is verifying the existence of a plan (i.e., a correct sequence of configuration actions like component instantiation, binding, or internal state changes) that, given a universe of available components and an initial component configuration, leads to a configuration where a target component is in a given state.

As an example, consider the task of reconfiguring a system setting up a *MySQL master-slave replication* avoiding the downtime of an existing legacy MySQL database. Reconfigurations of this kind are frequent in practice, and are nowadays performed by system administrators who execute reconfiguration receipts that are part of their know-how. According to the Aeolus model, the problem can be formalised as follows. The involved components are two distinct database instances, one in master mode and one in slave mode. We assume to start from a configuration with only one legacy running instance, that will become the master in the new configuration. To activate the slave, a *dump* of the data stored in the master is needed. Moreover, the master has to authorise the slave. This is a circular dependency that is resolved by forcing a precise order in which the reconfiguration actions can be performed: the master first requires authentication of the slave that, subsequently, requires the dump from the master.

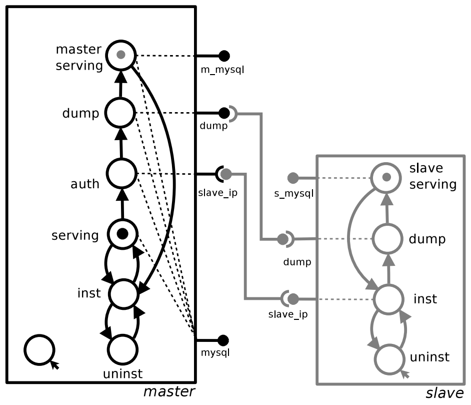


Fig. 1: MySQL master-slave instances (in black the initial configuration, in grey the parts added by the reconfiguration and the new state of the master).

In Fig. 1, following the Aeolus model, we depict how to configure MySQL components as master or as slave. We assume the master component to be a legacy one, meaning that it can not be created from scratch but has to be used as deployed in the initial configuration. This is technically obtained setting a dummy state with no outgoing transitions as the initial one. In this way, no newly legacy component could be generated and moved in a state that is different from the dummy one. Apart from the initial dummy state, the master component has 5 more states. The uninst state is followed by inst and serving. In serving, the master activates the provide-port mysql used by the clients to access the database service. When replication is needed, in order to enter the final master serving state, it first traverses the state auth that requires the IP address from the slave, and the state dump to provide the dump to the slave. The slave has instead 4 states, an initial uninst state and 3 states which complement those of the master during the replication process.

The formal definition of the Aeolus model is based on the notion of *component type*, used to specify the behaviour of a particular kind of component. In the following, $\mathcal{I}$ denotes the set of port names and $\mathcal{Z}$ the set of components.

Definition 1 (Component type). *The set Γ of* component types *ranged over by $\mathcal{T}, \mathcal{T}_1, \mathcal{T}_2, \ldots$ contains 4-tuples $\langle Q, q_0, T, D\rangle$ where:*

- *Q is a finite set of states containing the initial state q_0;*
- *$T \subseteq Q \times Q$ is the set of* transitions*;*
- *D is a function from Q to a pair $\langle \mathbf{P}, \mathbf{R}\rangle$ of port names (i.e., $\mathbf{P}, \mathbf{R} \subseteq \mathcal{I}$) indicating the provide-ports and require-ports that each state activates. We assume that the initial state q_0 has no requirements (i.e., $D(q_0) = \langle \mathbf{P}, \emptyset\rangle$).*

Configurations describe systems composed by components and their bindings. Many-to-many bindings connect components providing a functionality with components requiring it. Each component has a unique identifier, taken from the set $\mathcal{Z}$. A configuration, ranged over by $\mathcal{C}_1, \mathcal{C}_2, \ldots$, is given by a set of available component types, a set of component instances in some state, and a set of bindings.

Definition 2 (Configuration). *A* configuration *$\mathcal{C}$ is a quadruple $\langle U, Z, S, B\rangle$ where:*

- *$U \subseteq \Gamma$ is the finite* universe *of the available component types;*
- *$Z \subseteq \mathcal{Z}$ is the set of the currently deployed* components*;*
- *S is the component* state description*, i.e., a function that associates to components in Z a pair $\langle \mathcal{T}, q\rangle$ where $\mathcal{T} \in U$ is a component type $\langle Q, q_0, T, D\rangle$, and $q \in Q$ is the current component state;*
- *$B \subseteq \mathcal{I} \times Z \times Z$ is the set of* bindings*, namely 3-tuples composed by a port, the component that provides that port, and the component that requires it; we assume that the two components are distinct.*

Notation. We write $\mathcal{C}[z]$ as a lookup operation that retrieves the pair $\langle \mathcal{T}, q\rangle = S(z)$, where $\mathcal{C} = \langle U, Z, S, B\rangle$. On such a pair we then use the postfix projection operators `.type` and `.state` to retrieve $\mathcal{T}$ and q, respectively. Similarly, given a component type $\langle Q, q_0, T, D\rangle$, we use projections to decompose it: `.states`, `.init`, and `.trans` return the first three elements; $.\mathbf{P}(q)$ and $.\mathbf{R}(q)$ return the two elements of the $D(q)$ tuple. Moreover, we use `.prov` (resp. `.req`) to denote the union of all the provide-ports (resp. require-ports) of the states in Q. When there is no ambiguity we take the liberty to apply the component type projections to

$\langle \mathcal{T}, q\rangle$ pairs. *Example*: $\mathcal{C}[z].\mathbf{R}(q)$ stands for the require-ports of component z in configuration $\mathcal{C}$ when it is in state q.

As formalised below, a configuration is correct if all the active require-ports are bound to active provide-ports.

Definition 3 (Correctness). *Let us consider the configuration $\mathcal{C} = \langle U, Z, S, B\rangle$.*

We write $\mathcal{C} \models_{req} (z, r)$ to indicate that the require-port of component z, with port r, is bound to an active port providing r, i.e., there exists a component $z' \in Z \setminus \{z\}$ such that $\langle r, z', z\rangle \in B$, $\mathcal{C}[z'] = \langle \mathcal{T}', q'\rangle$ and r is in $\mathcal{T}'.\mathbf{P}(q')$.

The configuration $\mathcal{C}$ is correct *if for every component $z \in Z$ with $S(z) = \langle \mathcal{T}, q\rangle$ we have that $\mathcal{C} \models_{req} (z, r)$ for every $r \in \mathcal{T}.\mathbf{R}(q)$.*

In Aeolus configurations evolve by means of (deployment) actions.

Definition 4 (Actions). *The set $\mathcal{A}$ contains the following actions:*

- *$stateChange(z, q, q')$ changes the state of the component $z \in \mathcal{Z}$ from q to q';*
- *$bind(r, z_1, z_2)$ creates a binding between the provide-port $r \in \mathcal{I}$ of the component z_1 and the require-port r of z_2 ($z_1, z_2 \in \mathcal{Z}$);*
- *$unbind(r, z_1, z_2)$ deletes the binding between the provide-port $r \in \mathcal{I}$ of the component z_1 and the require-port r of z_2 ($z_1, z_2 \in \mathcal{Z}$);*
- *$new(z : \mathcal{T})$ creates a new component of type $\mathcal{T}$ in its initial state. The new component is identified by a unique and fresh identifier $z \in \mathcal{Z}$;*
- *$del(z)$ deletes the component $z \in \mathcal{Z}$.*

The execution of actions is formalised by means of a labelled transition system on configurations, which uses actions as labels.

Definition 5 (Reconfigurations). *Reconfigurations are denoted by transitions $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}'$ meaning that the execution of $\alpha \in \mathcal{A}$ on the configuration $\mathcal{C}$ produces a new configuration $\mathcal{C}'$. The transitions from a configuration $\mathcal{C} = \langle U, Z, S, B\rangle$ are defined as follows:*

$$\mathcal{C} \xrightarrow{stateChange(z,q,q')} \langle U, Z, S', B\rangle \quad \text{if } \mathcal{C}[z].\mathtt{state} = q \text{ and } (q, q') \in \mathcal{C}[z].\mathtt{trans} \text{ and } S'(z') = \begin{cases} \langle \mathcal{C}[z].\mathtt{type}, q'\rangle & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases}$$

$$\mathcal{C} \xrightarrow{bind(r,z_1,z_2)} \langle U, Z, S, B \cup \langle r, z_1, z_2\rangle\rangle \quad \text{if } \langle r, z_1, z_2\rangle \notin B \text{ and } r \in \mathcal{C}[z_1].\mathtt{prov} \cap \mathcal{C}[z_2].\mathtt{req}$$

$$\mathcal{C} \xrightarrow{unbind(r,z_1,z_2)} \langle U, Z, S, B \setminus \langle r, z_1, z_2\rangle\rangle \quad \text{if } \langle r, z_1, z_2\rangle \in B$$

$$\mathcal{C} \xrightarrow{new(z:\mathcal{T})} \langle U, Z \cup \{z\}, S', B\rangle \quad \text{if } z \notin Z,\ \mathcal{T} \in U \text{ and } S'(z') = \begin{cases} \langle \mathcal{T}, \mathcal{T}.\mathtt{init}\rangle & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases}$$

$$\mathcal{C} \xrightarrow{del(z)} \langle U, Z \setminus \{z\}, S', B'\rangle \quad \text{if } S'(z') = \begin{cases} \bot & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases} \text{ and } B' = \{\langle r, z_1, z_2\rangle \in B \mid z \notin \{z_1, z_2\}\}$$

A *deployment plan* is simply a sequence of actions that transform a correct configuration without violating correctness along the way.

Definition 6 (Deployment plan). *A* deployment plan P *from a correct configuration $\mathcal{C}_0$ is a sequence of actions $\alpha_1, \ldots, \alpha_m$ s.t. there exists $\mathcal{C}_1, \ldots, \mathcal{C}_m$ correct configurations s.t. $\mathcal{C}_{i-1} \xrightarrow{\alpha_i} \mathcal{C}_i$.*

In the following, exploiting the fact that reconfigurations are deterministic, we denote the deployment plan $\alpha_1, \ldots, \alpha_m$ from $\mathcal{C}_0$ also with the sequence of reconfigurations steps $\mathcal{C}_0 \xrightarrow{\alpha_1} \mathcal{C}_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} \mathcal{C}_m$.

We now have all the ingredients to define the *reconfiguration problem*, that is our main concern: given a universe of component types and an initial configuration, we want to know whether and how it is possible to deploy at least one component of a given component type $\mathcal{T}$ in a given state q.

Definition 7 (Reconfiguration problem). *The* reconfiguration problem *has as input a universe U of component types, an initial correct configuration $\mathcal{C}_0$, a component type $\mathcal{T}_t$, and a target state q_t. The output is* **yes** *if there exists a deployment plan* $\mathsf{P} = \mathcal{C}_0 \xrightarrow{\alpha_1} \mathcal{C}_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} \mathcal{C}_m$ *s.t.* $\mathcal{C}_m[z] = \langle \mathcal{T}_t, q_t \rangle$*, for some component z in $\mathcal{C}_m$. Otherwise, it returns* **no***, stating that no such plan exists.*

As an example, considering Fig. 1, we can see that there are deployment plans that lead from the initial configuration (in black) to the final MySQL master-slave replication configuration. For instance, such a plan could start with the creation of the slave instance, followed by a state change to the inst state and the creation of a binding between the ports slave_ip of the two components. At this point, the master component can perform two state changes, reaching the dump state. Then, after another binding is established between the dump ports, the slave can be moved to its serving state by performing two state changes. Finally, the master can enter in the master serving state by performing a state change. Note that every action in the deployment plan will correspond to one or more concrete instructions. For instance, the state change from the serving to the auth state in the master corresponds to issue the command `grant replication slave on *.* to user@'slave_ip'`.

The addition of a dummy initial state to define the master component captures its legacy nature. Indeed, since no other state of the master component is reachable from the initial one, no component created from scratch can provide the same functionalities of the deployed master. For this reason, only the master component present in the initial configuration can be used to reach the target.

Notice that the restriction to consider one target state only in the definition of the reconfiguration problem is not limiting: one can require several target pairs $\langle \mathcal{T}_t, q_t \rangle$ by adding dummy provide-ports enabled only by the components of type $\mathcal{T}_t$ in state q_t and a dummy target component that requires all such provides. For instance, Fig. 2 depicts the dummy target component that in inst state requires both an active master and an active slave as needed in the MySQL master-slave reconfiguration discussed above.

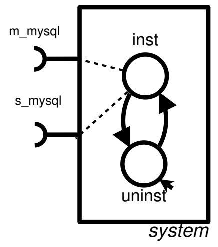


Fig. 2: Target

3 Solving the Reconfiguration Problem

In this section we present a nondeterministic polynomial space algorithm that resolves the reconfiguration problem, thus the problem is proved to be in PSpace (as a consequence of the Savitch's theorem [22] stating the equivalence between NPSpace and PSpace). The idea is to perform a nondeterministic forward exploration of the reachable configurations. This visit could be in principle arbitrarily long because infinitely many different configurations could be potentially reached. The main result that we prove in this section is that it is sufficient to consider a bounded amount of possibly reachable *abstract* configurations. In abstract configurations the bindings are not considered, but only the component type and state of the components are taken into account. Moreover, in abstract configurations, only the components present in the initial configuration are precisely represented, while for all the other components that are dynamically created, it is only considered the presence or absence of instances of components of type $\mathcal{T}$ in state q, thus abstracting away from their precise number.

In order to abstract away from the bindings and consider only the component types and states, we define the following equivalence among configurations.

Definition 8 (Configuration equivalence). *Two configurations $\langle U, Z, S, B\rangle$ and $\langle U, Z', S', B'\rangle$ are equivalent ($\langle U, Z, S, B\rangle \equiv \langle U, Z', S', B'\rangle$) iff there exists a bijective function ρ from Z to Z' s.t. $S(z) = S'(\rho(z))$ for every $z \in Z$.*

The research of the existence of the deployment plan is done on abstract configurations where bindings are not considered. We now show that this is not restrictive because every plan has a corresponding *normalised* plan where unbinding actions are absent and binding actions are generated as soon as possible.

Definition 9 (Normalised deployment plan). *A deployment plan $\mathsf{P} = \mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} \mathcal{C}_m$ is normalised iff:*

- *it does not contain unbind actions,*
- *if $\mathcal{C}_i$ for $i \in [1, m-1]$ can be extended with a bind action then $\xrightarrow{\alpha_{i+1}}$ is a bind action,*
- *$\mathcal{C}_m$ cannot be extended with a bind action.*

Lemma 1. *Given a deployment plan $\mathsf{P} = \mathcal{C}_0 \xrightarrow{\alpha_1} \mathcal{C}_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} \mathcal{C}_m$ there exists a normalised deployment plan $\mathsf{P}' = \mathcal{C}_0 \xrightarrow{\alpha'_1} \mathcal{C}'_1 \xrightarrow{\alpha'_2} \cdots \xrightarrow{\alpha'_n} \mathcal{C}_n$ such that $\mathcal{C}_n \equiv \mathcal{C}_m$.*

In the remainder of the section, we assume a given universe U of component types; so we can consider that the set of distinct component type and state pairs $\langle \mathcal{T}, q\rangle$ is finite. Let k be its cardinality. Moreover, we assume a given initial configuration $\mathcal{C}_0$ having the initial set of components Z_0.

We are now ready to define our abstractions $\mathcal{B}$ consisting of pairs of functions $\langle \mathcal{B}_i, \mathcal{B}_c\rangle$. Components are divided into two groups, those that were present in the *initial* configuration and those that were dynamically *created*: the first ones are precisely counted by the function $\mathcal{B}_i$, while for the second ones only the presence of a component type and state pair $\langle \mathcal{T}, q\rangle$ is checked by the function $\mathcal{B}_c$.

Definition 10 (Abstract configuration). *An abstract configuration $\mathcal{B}$ is a pair of functions $\langle \mathcal{B}_i, \mathcal{B}_c\rangle$ that associate to every pair $\langle \mathcal{T}, q\rangle$ respectively a natural number and a boolean value.*

It is immediate to see that (given a universe U of component types and an initial set Z_0 of components) the set of possible abstract configurations is finite: both functions have a domain bound by k, $\mathcal{B}_c$ is a boolean function, and the sum of the values in the codomain of $\mathcal{B}_i$ is bound by $|Z_0|$, i.e., the number of initial components, because such components can only be destroyed and not created.

A concretisation of an abstract configuration $\langle \mathcal{B}_i, \mathcal{B}_c \rangle$ is defined w.r.t. a set of initial components Z. These components occur according to the component type/state pairs counted by $\mathcal{B}_i$, while the other components satisfy the presence/absence indication of the boolean function $\mathcal{B}_c$. In the definition of concretisation we use the following notations: $\mathcal{C}^{\#}_{\langle \mathcal{T}, q \rangle}(Z)$ is the number of components in Z of type $\mathcal{T}$ in state q in the configuration $\mathcal{C}$, while $Z - Z'$ is the set difference between two sets of components Z and Z'.

Definition 11 (Concretisation). *Given an abstract configuration $\mathcal{B} = \langle \mathcal{B}_i, \mathcal{B}_c \rangle$ and a set of components Z we say that a correct configuration $\mathcal{C} = \langle U, Z', S, B \rangle$ is one concretisation of $\mathcal{B}$ w.r.t. Z if the following hold:*

- $\mathcal{B}_i(\langle \mathcal{T}, q \rangle) = \mathcal{C}^{\#}_{\langle \mathcal{T}, q \rangle}(Z)$;
- *if* $\neg \mathcal{B}_c(\langle \mathcal{T}, q \rangle)$ *then* $\mathcal{C}^{\#}_{\langle \mathcal{T}, q \rangle}(Z' - Z) = 0$;
- *if* $\mathcal{B}_c(\langle \mathcal{T}, q \rangle)$ *then* $\mathcal{C}^{\#}_{\langle \mathcal{T}, q \rangle}(Z' - Z) > 0$.

We denote with $\gamma(\mathcal{B}, Z)$ the set of concretisations of $\mathcal{B}$ w.r.t. Z. We say that an abstract configuration $\mathcal{B}$ is correct w.r.t. Z if it has at least one concretisation (formally $\gamma(\mathcal{B}, Z) \neq \emptyset$).

In the following, we usually consider concretisations w.r.t. the initial set of components Z_0, and we simply use $\gamma(\mathcal{B})$ to denote $\gamma(\mathcal{B}, Z_0)$.

We now define the notion of deployment plan on abstract configurations and formalise its correspondence with *concrete* normalised plans.

Definition 12 (Abstract deployment plan). *We write $\mathcal{B} \rightarrow \mathcal{B}'$ with $\mathcal{B} \neq \mathcal{B}'$ if there exists $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}'$ for some $\mathcal{C} \in \gamma(\mathcal{B})$ and $\mathcal{C}' \in \gamma(\mathcal{B}')$.*

A first lemma proves that each normalised deployment plan has a corresponding abstract version.

Lemma 2. *Given a normalised deployment plan $\mathsf{P} = \mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} \mathcal{C}_m$ there is an abstract deployment plan $\mathcal{B}_0 \rightarrow \cdots \rightarrow \mathcal{B}_n$ s.t. $\mathcal{C}_0 \in \gamma(\mathcal{B}_0)$ and $\mathcal{C}_m \in \gamma(\mathcal{B}_n)$.*

The opposite correspondence (each abstract plan has at least one corresponding normalised *concrete* plan) is more complex to be formalised and proved. The intuition is that, given an abstract configuration $\mathcal{B} = \langle \mathcal{B}_i, \mathcal{B}_c \rangle$ that can be reached by an abstract plan, there exist normalised deployment plans able to reconfigure exactly the initial components as indicated by $\mathcal{B}_i$, and deploy an arbitrary number of instances of other components in the type and state indicated by the boolean function $\mathcal{B}_c$.

Lemma 3. *Given a correct configuration $\mathcal{C}_0$ that cannot be extended with bind actions and an abstract deployment plan $\mathcal{B}_0 \to \cdots \to \mathcal{B}_n = \langle \mathcal{B}_i, \mathcal{B}_c \rangle$ such that $\mathcal{C}_0 \in \gamma(\mathcal{B}_0)$ then there is a normalised deployment plan $\mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{m-1}} \mathcal{C}_m$ s.t.:*

- *$\mathcal{C}_m \in \gamma(\mathcal{B}_n)$;*
- *for all natural numbers $j_{\langle \mathcal{T}, q \rangle} > 0$, for every component type $\mathcal{T}$ and state q such that $\mathcal{B}_c(\langle \mathcal{T}, q \rangle)$, then $\mathcal{C}_m{}^{\#}_{\langle \mathcal{T}, q \rangle}(Z_m - Z_0) = j_{\langle \mathcal{T}, q \rangle}$ where Z_m and Z_0 are the components of $\mathcal{C}_m$ and $\mathcal{C}_0$ respectively.*

Algorithm 1 Nondeterministic check for $\mathcal{C}_0 = \langle U, Z_0, S, B \rangle$ and target $\mathcal{T}_t, q_t$

```
for all $\langle \mathcal{T}, q \rangle$ pairs in the universe $U$ do
    $\mathcal{B}_i(\langle \mathcal{T}, q \rangle) = \mathcal{C}^{\#}_{\langle \mathcal{T}, q \rangle}(Z_0)$
    $\mathcal{B}_c(\langle \mathcal{T}, q \rangle) = False$
$counter = 0$
while $counter \leq |Z_0|^k * 2^k$ do                ▷ k is the number of $\langle \mathcal{T}, q \rangle$ pairs in $U$
    guess $\mathcal{B}'_i, \mathcal{B}'_c$
    if $\langle \mathcal{B}_i, \mathcal{B}_c \rangle \not\to \langle \mathcal{B}'_i, \mathcal{B}'_c \rangle$ then return Failure
    if $\mathcal{B}'_i(\mathcal{T}_t, q_t) > 0$ or $\mathcal{B}'_c(\mathcal{T}_t, q_t)$ then return Success
    $counter = counter + 1$; $\mathcal{B}_i = \mathcal{B}'_i$; $\mathcal{B}_c = \mathcal{B}'_c$
return Failure
```

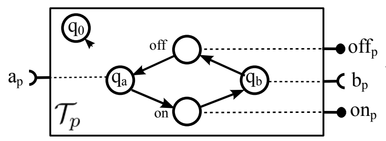
In order to check if a solution to the reconfiguration problem exists, it is possible to consider all the possible abstract plans. This can be done using the nondeterministic Algorithm 1. Starting from the abstract representation $\langle \mathcal{B}_i, \mathcal{B}_c \rangle$ of the initial configuration $\mathcal{C}_0$, it performs a nondeterministic exploration of the reachable abstract configurations until either a configuration containing the target $\langle \mathcal{T}_t, q_t \rangle$ is reached or at least $K = |Z_0|^k * 2^k$ abstract steps have been considered, where $|Z_0|$ is the quantity of components of the initial configuration and k is the number of different $\langle \mathcal{T}, q \rangle$ pairs in the universe U. K is an upper bound to the number of different abstract configurations: $|Z_0|^k$ is an upper bound to the different combinations of states for the initially available components, while 2^k is the number of possible sets of $\langle \mathcal{T}, q \rangle$ pairs.

Assuming n the size of the input we have that $|Z_0| \leq n, k \leq n$ and therefore all the variables of the nondeterministic Algorithm 1 can be encoded in $O(n \log(n))$ space. For this reason (and for Savitch's theorem [22]) we can conclude that the reconfiguration problem is in PSpace.

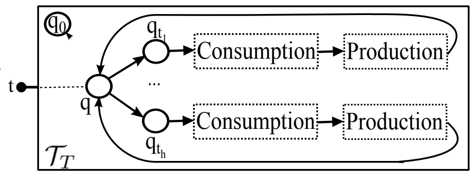
Theorem 1. *The reconfiguration problem is PSpace.*

4 The Reconfiguration Problem is PSpace-hard

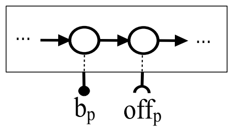
PSpace-hardness of the reconfiguration problem is proved by reduction from the reachability problem in 1-safe Petri nets, which is indeed known to be a PSpace-hard problem [3]. We start with some background on Petri nets.



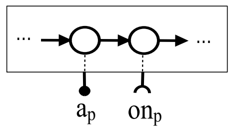
(a) Token in place p.



(b) Transitions component.



(c) Consumption from p.



(d) Production in p.

Fig. 3: 1-safe Petri net encoding

A *Petri net* is a tuple $N = (P, T, \boldsymbol{m_0})$, where P and T are finite sets of *places* and *transitions*, respectively. A finite multiset over the set P of places is called a *marking*, and $\boldsymbol{m_0}$ is the initial marking. Given a marking $\boldsymbol{m}$ and a place p, we say that the place p contains a number of *tokens* equal to the number of instances of p in $\boldsymbol{m}$. A transition $t \in T$ is a pair of markings denoted with ${}^\bullet t$ and $t^\bullet$. A transition t can fire in the marking $\boldsymbol{m}$ if ${}^\bullet t \subseteq \boldsymbol{m}$ (where $\subseteq$ is multiset inclusion); upon transition firing the new marking of the net becomes $\boldsymbol{n} = (\boldsymbol{m} \setminus {}^\bullet t) \uplus t^\bullet$ (where $\setminus$ and $\uplus$ are the difference and union operators for multisets, respectively). This is written as $\boldsymbol{m} \mapsto \boldsymbol{n}$. We use $\mapsto^*$ to denote the reflexive and transitive closure of $\mapsto$. We say that $\boldsymbol{m'}$ is *reachable from* $\boldsymbol{m}$ if $\boldsymbol{m} \mapsto^* \boldsymbol{m'}$. A Petri net P is 1-safe if in every reachable marking every place has at most one token. Reachability of a specific marking $\boldsymbol{m_t}$ from the initial marking $\boldsymbol{m_0}$ is PSpace-complete for 1-safe nets [3].

We now consider a given 1-safe Petri net $N = \langle P, T, \boldsymbol{m_0} \rangle$ and discuss how to encode it in Aeolus component types. We will use two types of legacy components: one modelling the places and one for the transitions. The simplest component type, denoted with $\mathcal{T}_p$ and depicted in Fig. 3a, is the one used to model a place $p \in P$. Namely, a place p is encoded as one instance of $\mathcal{T}_p$. A token is present in p if the component of type $\mathcal{T}_p$ is in the *on* state. There could be just one of these components deployed simultaneously. This can be obtained simply adding this component to the initial configuration in the *on* or *off* state, according to the initial marking, and make these two states non reachable from the initial state q_0. The token could be created starting from the *off* state following a protocol consisting of providing the port a_p and then requiring the port on_p. Symmetrically, a token can be removed by providing the port b_p and then requiring the port off_p. The component provides the port on_p when it is in the *on* state, the port off_p when it is in the *off* state.

The transitions in T can be represented with a single component of type $\mathcal{T}_T$ depicted in Fig. 3b. The uniqueness of this component is guaranteed, as done for

$\mathcal{T}_p$, by adding it to the initial configuration and forbidding outgoing transitions from the initial state q_0. This component is assumed to be present in the initial configuration in state q. From this state it can nondeterministically select one transition t to fire, by entering a corresponding q_t state. The subsequent state changes can be divided into two phases: consumption and production. These phases respectively model the consumption of tokens from the places in the preset of t and the production of tokens in the places in the postset of t. The consumption and production of tokens have been already discussed above: consumption (see Fig. 3c) is obtained by providing and requiring the ports b_p and off_p, production (see Fig. 3d) by providing and requiring the ports a_p and on_p.

We now consider a marking $\boldsymbol{m_t}$ of the 1-safe Petri net N. We can check whether $\boldsymbol{m_t}$ is reachable in N by considering the following Aeolus reconfiguration problem. The initial configuration consists of an instance of $\mathcal{T}_T$, in state q, plus a component of type $\mathcal{T}_p$ for every place p, in *on* or *off* depending on the initial marking $\boldsymbol{m_0}$. The target to be considered consists of a configuration in which a port on_p is active for all places $p \in \boldsymbol{m_t}$, a port off_p is active for all places $p \notin \boldsymbol{m_t}$ and the port t is active indicating that no transition is currently in execution. Checking these requirements can be easily done, as explained in Section 2, by adding a dummy component having a target state requiring all the ports as explained above. Hence, we have the following.

Theorem 2. *The reconfiguration problem is PSpace-hard.*

5 Related Work and Conclusions

To the best of our knowledge, there is no work that formally studies the complexity of automatic reconfiguration of component systems. A significant part of the related literature focuses on the problem of dynamic re-allocation of resources (see, e.g., [4,12,23]). Other works focus on the nature of the reconfiguration problem, like in [25] where a classification of the reconfiguration problems is made based on its causes, namely failures, system updates, and user requests. This work, however, does not consider the complexity of establishing the reconfiguration steps. Formal methods have been used to study reconfiguration problems as, for instance, in [16] where graph transformations and model checking are used to reason about dynamically changing component connectors. The focus in this case, however, is on proving properties of the reconfigured system.

Different tools to compute the (optimal) final configuration exist. For instance, Zephyrus [5] takes as input a description of the system in Aeolus and, given an initial configuration, computes the best final configuration satisfying the application manager requests. Similarly, in [19] a prediction-based online-approach is proposed to find optimal reconfiguration policies, in [10] a genetic-based algorithm is used to support the migration and deployment of enterprise software with their reconfiguration policies, in [14] a constraint-based approach is used to propose an optimal allocation of virtual machines and applications on servers, and in [24] integer linear programming methods are used to find energy efficient optimal final configurations. All these approaches just focus on the target configuration to reach without computing the deployment steps.

AI Planning Technologies [11] have been used to generate automatically the actions to reconfigure a system [2, 13]. However, since the classical planning problem is PSpace [1], these techniques have scalability issues. Conversely, tools like Metis [17,18] or Engage [9] are able to compute the deployment steps needed to reach a target configuration but in simplified contexts: Metis imposes empty initial configurations while Engage forbids circular dependencies.

In this work we proved that extending these tools to deal also with reconfigurations may be too computationally expensive. Indeed, PSpace-completeness means that there are at least some cases where solving a reconfiguration problem requires a huge computational effort. For instance, our hardness proof shows that this can happen in the presence of legacy components that can not be recreated from scratch and may be required to perform cycles of deployment actions.

As a future work we plan to investigate limitations to be imposed to the Aeolus model (e.g., limiting the shape of the automata describing the components lifecycle) in order to have more efficient solutions for the reconfiguration problem. Another approach could be to relax completeness, by designing algorithms that could give negative answers even if a solution exists. Along this direction, one promising approach is the use of heuristics to guide the exploration of the (abstract) search space used by our nondeterministic algorithm in Section 3.

References

1. T. Bylander. Complexity Results for Planning. In *IJCAI*, 1991.
2. M. Chen, P. Poizat, and Y. Yan. Adaptive Composition and QoS Optimization of Conversational Services Through Graph Planning Encoding. In *Web Services Foundations*. 2014.
3. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theor. Comput. Sci.*, 1995.
4. H. W. Choi, H. Kwak, A. Sohn, and K. Chung. Autonomous learning for efficient resource utilization of dynamic VM migration. In *ICS*, 2008.
5. R. D. Cosmo, M. Lienhardt, R. Treinen, S. Zacchiroli, J. Zwolakowski, A. Eiche, and A. Agahi. Automated synthesis and deployment of cloud applications. In *ASE*, 2014.
6. R. D. Cosmo, J. Mauro, S. Zacchiroli, and G. Zavattaro. Aeolus: A component model for the cloud. *Inf. Comput.*, 2014.
7. R. Di Cosmo, A. Eiche, J. Mauro, G. Zavattaro, S. Zacchiroli, and J. Zwolakowski. Automatic Deployment of Software Components in the Cloud with the Aeolus Blender. Technical report, Inria Sophia Antipolis, 2015.
8. R. Di Cosmo, J. Mauro, S. Zacchiroli, and G. Zavattaro. Component Reconfiguration in the Presence of Conflicts. In *ICALP*, 2013.
9. J. Fischer, R. Majumdar, and S. Esmaeilsabzali. Engage: a deployment management system. In *PLDI*, 2012.
10. S. Frey, F. Fittkau, and W. Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *ICSE*, 2013.
11. M. Ghallab, D. S. Nau, and P. Traverso. *Automated planning - Theory and Practice.* Elsevier, 2004.

12. D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *DSN*, 2008.
13. H. Herry, P. Anderson, and G. Wickler. Automated Planning for Configuration Changes. In *LISA*, 2011.
14. J. A. Hewson, P. Anderson, and A. D. Gordon. A Declarative Approach to Automated Configuration. In *LISA*, 2012.
15. Juju, DevOps distilled. `https://juju.ubuntu.com/`.
16. C. Krause. *Reconfigurable Component Connectors*. PhD thesis, Univ. Leiden, 2011.
17. T. A. Lascu, J. Mauro, and G. Zavattaro. A Planning Tool Supporting the Deployment of Cloud Applications. In *ICTAI*, 2013.
18. T. A. Lascu, J. Mauro, and G. Zavattaro. Automatic Component Deployment in the Presence of Circular Dependencies. In *FACS*, 2013.
19. H. Mi, H. Wang, G. Yin, Y. Zhou, D. xi Shi, and L. Yuan. Online Self-Reconfiguration with Performance Guarantee for Energy-Efficient Large-Scale Cloud Computing Data Centers. In *SCC*, 2010.
20. K. Morris. Immutableserver. `http://martinfowler.com/bliki/ImmutableServer.html`, 2013.
21. OASIS. Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. `http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html`.
22. W. J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *J. Comput. Syst. Sci.*, 1970.
23. J. Stoess, C. Lang, and F. Bellosa. Energy Management for Hypervisor-Based Virtual Machines. In *USENIX Annual Tech. Conf.*, 2007.
24. P. N. Tran, L. Casucci, and A. Timm-Giel. Optimal mapping of virtual networks considering reactive reconfiguration. In *CLOUDNET*, 2012.
25. S. Wang, F. Du, X. Li, Y. Li, and X. Han. Research on dynamic reconfiguration technology of cloud computing virtual services. In *CCIS*, 2011.

A Proofs of Section 3

Lemma 1. *Given a deployment plan* $\mathsf{P} = \mathcal{C}_0 \xrightarrow{\alpha_1} \mathcal{C}_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} \mathcal{C}_m$ *there exists a normalised deployment plan* $\mathsf{P}' = \mathcal{C}_0 \xrightarrow{\alpha'_1} \mathcal{C}'_1 \xrightarrow{\alpha'_2} \cdots \xrightarrow{\alpha'_n} \mathcal{C}_n$ *such that* $\mathcal{C}_n \equiv \mathcal{C}_m$*.*

Proof. Considering the Aeolus fragment defined in Section 2, it is always possible to perform a bind action $\mathit{bind}(r, z_1, z_2)$ if the two components z_1 and z_2 exist and the provide-port r of z_1 is not yet bound to the require-port r of z_2.[2] Hence, bindings do not require that the connected port is active and can be performed as soon as the two components are created. For this reason, the normalised plan P' can be easily obtained by

– removing from P the unbind and bind actions;
– adding as soon as possible in P' the binding actions between two complementary ports that are not yet bound (i.e., bind all the complementary ports after every component creation or at the outset of the plan).

□

Lemma 2. *Given a normalised deployment plan* $\mathsf{P} = \mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} \mathcal{C}_m$ *there is an abstract deployment plan* $\mathcal{B}_0 \to \cdots \to \mathcal{B}_n$ *s.t.* $\mathcal{C}_0 \in \gamma(\mathcal{B}_0)$ *and* $\mathcal{C}_m \in \gamma(\mathcal{B}_n)$*.*

Proof. The abstract deployment plan $\mathcal{B}_0 \to \cdots \to \mathcal{B}_n$ can be constructed simply by taking the sequence of the abstract versions of the transitions $\mathcal{C}_i \xrightarrow{\alpha_{i+1}} \mathcal{C}_{i+1}$ in P such that the abstraction of $\mathcal{C}_i$ is different from the abstraction of $\mathcal{C}_{i+1}$. □

Lemma 3. *Given a correct configuration* $\mathcal{C}_0$ *that cannot be extended with bind actions and an abstract deployment plan* $\mathcal{B}_0 \to \cdots \to \mathcal{B}_n = \langle \mathcal{B}_i, \mathcal{B}_c \rangle$ *such that* $\mathcal{C}_0 \in \gamma(\mathcal{B}_0)$ *then there is a normalised deployment plan* $\mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{m-1}} \mathcal{C}_m$ *s.t.:*

– $\mathcal{C}_m \in \gamma(\mathcal{B}_n)$*;*
– *for all natural numbers* $j_{\langle \mathcal{T}, q \rangle} > 0$*, for every component type* $\mathcal{T}$ *and state* q *such that* $\mathcal{B}_c(\langle \mathcal{T}, q \rangle)$*, then* $\mathcal{C}_m{}^{\#}_{\langle \mathcal{T}, q \rangle}(Z_m - Z_0) = j_{\langle \mathcal{T}, q \rangle}$ *where* Z_m *and* Z_0 *are the components of* $\mathcal{C}_m$ *and* $\mathcal{C}_0$ *respectively.*

Proof. The proof of the lemma is by induction on the length of the abstract deployment plan.

Base case. If the length is 0 then $\mathcal{B}_0 = \mathcal{B}_n$. The first item holds because $\mathcal{C}_0 \in \gamma(\mathcal{B}_0) = \gamma(\mathcal{B}_n)$, while the second one trivially holds because $\neg \mathcal{B}_c(\langle \mathcal{T}, q \rangle)$ for every component type/state pair $\langle \mathcal{T}, q \rangle$. Indeed, since in the initial configuration no component is created (no new actions have been performed yet), $\mathcal{B}_c(\langle \mathcal{T}, q \rangle)$ is always false.

Inductive case. Consider the transition $\mathcal{B}_{n-1} \to \mathcal{B}_n$ and assume that $\mathcal{B}_{n-1} = \langle \mathcal{B}'_i, \mathcal{B}'_c \rangle$. We have by definition that there exists an action $\xrightarrow{\alpha}$ from a concretisation of $\mathcal{B}_{n-1}$ to a concretisation of $\mathcal{B}_n$. Since $\mathcal{B}_{n-1} \neq \mathcal{B}_n$, $\xrightarrow{\alpha}$ can just be a delete, create, or state change action. Now we have the following cases:

[2] Note that this holds because we consider a fragment of Aeolus that does not support capacity constraints to limit the amount of bindings connected to a provide-port.

- α *involves a component* z *in* Z_0. Then by inductive hypothesis there exists a normalised deployment plan $\mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_l} \mathcal{C}_l$ s.t. $\mathcal{C}_l \in \gamma(\mathcal{B}_{n-1})$ and if $\mathcal{B}'_c(\langle \mathcal{T}, q\rangle)$ then ${\mathcal{C}_l}^{\#}_{\langle \mathcal{T},q\rangle}(Z_l - Z_0) = j_{\langle \mathcal{T},q\rangle}$ where Z_l are the components of $\mathcal{C}_l$. Since $\mathcal{C}_l$ is a concretisation of $\mathcal{B}_{n-1}$ and α involves the initial component z we have that z is still present in $\mathcal{C}_l$. Hence, from $\mathcal{C}_l$ the α action can be performed leading to a correct configuration that is a concretisation of $\mathcal{B}_n$. Since in this transition only components in Z_0 are involved, $\mathcal{B}_c = \mathcal{B}'_c$ and then the thesis.
- α *is a create action* $new(z : \mathcal{T}')$ *and* $z \notin Z_0$. Then, due to Definition 12 requiring $\mathcal{B}'_c$ and $\mathcal{B}_c$ to be different, $\mathcal{B}'_c(\langle \mathcal{T}', \mathcal{T}'.\mathtt{init}\rangle)$ is false, $\mathcal{B}_c(\langle \mathcal{T}, q\rangle)$ is true if $\langle \mathcal{T}, q\rangle = \langle \mathcal{T}', \mathcal{T}'.\mathtt{init}\rangle$, equal to $\mathcal{B}'_c(\langle \mathcal{T}, q\rangle)$ otherwise. By inductive hypothesis there exists a normalised deployment plan $\mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_l} \mathcal{C}_l$ s.t. $\mathcal{C}_l \in \gamma(\mathcal{B}_{n-1})$ and if $\mathcal{B}'_c(\langle \mathcal{T}, q\rangle)$ then ${\mathcal{C}_l}^{\#}_{\langle \mathcal{T},q\rangle}(Z_l - Z_0) = j_{\langle \mathcal{T},q\rangle}$ where Z_l are the components of $\mathcal{C}_l$. From $\mathcal{C}_l$ is possible to perform $j_{\langle \mathcal{T}', \mathcal{T}'.\mathtt{init}\rangle}$ create actions, each one followed by all bind actions that can be performed to reach a configuration $\mathcal{C}_m$ with the required properties.
- α *is a delete action* $del(z)$ *that deletes a component of type* $\mathcal{T}'$ *in state* q' *and* $z \notin Z_0$. Then $\mathcal{B}'_c(\langle \mathcal{T}', q'\rangle)$ is true, $\mathcal{B}_c(\langle \mathcal{T}, q\rangle)$ is false if $\langle \mathcal{T}, q\rangle = \langle \mathcal{T}', q'\rangle$, equal to $\mathcal{B}'_c(\langle \mathcal{T}, q\rangle)$ otherwise. By inductive hypothesis there exists a normalised deployment plan $\mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_l} \mathcal{C}_l$ s.t. $\mathcal{C}_l \in \gamma(\mathcal{B}_{n-1})$ and if $\mathcal{B}'_c(\langle \mathcal{T}, q\rangle)$ then ${\mathcal{C}_l}^{\#}_{\langle \mathcal{T},q\rangle}(Z_l - Z_0) = j_{\langle \mathcal{T},q\rangle}$ where Z_l are the components of $\mathcal{C}_l$. Then from $\mathcal{C}_l$ it is possible to delete all the $j_{\langle \mathcal{T}', q'\rangle}$ components of type $\mathcal{T}'$ in state q' that are not in Z_0 and reach a configuration $\mathcal{C}_m$ with the required properties.
- α *is a state change* $stateChange(z, q', q'')$ *applied to a component type* $\mathcal{T}'$ *and* $z \notin Z_0$. We have that $\mathcal{B}'_c(\langle \mathcal{T}', q'\rangle) = \neg\mathcal{B}_c(\langle \mathcal{T}', q'\rangle)$ or $\mathcal{B}'_c(\langle \mathcal{T}', q''\rangle) = \neg\mathcal{B}_c(\langle \mathcal{T}', q''\rangle)$, and $\mathcal{B}'_c(\langle \mathcal{T}, q\rangle) = \mathcal{B}_c(\langle \mathcal{T}, q\rangle)$ otherwise. Moreover, we have that $\mathcal{B}'_c(\langle \mathcal{T}', q'\rangle)$ and $\mathcal{B}_c(\langle \mathcal{T}', q''\rangle)$ must be true.
  Now, if $\mathcal{B}'_c(\langle \mathcal{T}', q'\rangle) = \neg\mathcal{B}_c(\langle \mathcal{T}', q'\rangle)$ let us consider

$$j'_{\langle \mathcal{T},q\rangle} = \begin{cases} 1 & \text{if } \langle \mathcal{T}, q\rangle = \langle \mathcal{T}', q''\rangle \text{ and } \mathcal{B}'_c(\langle \mathcal{T}', q''\rangle) \\ j_{\langle \mathcal{T}', q''\rangle} & \text{if } \langle \mathcal{T}, q\rangle = \langle \mathcal{T}', q'\rangle \\ j_{\langle \mathcal{T},q\rangle} & \text{otherwise} \end{cases}$$

  By inductive hypothesis there exists a normalised deployment plan $\mathcal{C}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_l} \mathcal{C}_l$ s.t. $\mathcal{C}_l \in \gamma(\mathcal{B}_{n-1})$ and if $\mathcal{B}'_c(\langle \mathcal{T}, q\rangle)$ then ${\mathcal{C}_l}^{\#}_{\langle \mathcal{T},q\rangle}(Z_l - Z_0) = j'_{\langle \mathcal{T},q\rangle}$ where Z_l are the components of $\mathcal{C}_l$. Now, starting from $\mathcal{C}_l$, it is possible to perform $j'_{\langle \mathcal{T}', q'\rangle} = j_{\langle \mathcal{T}', q''\rangle}$ state changes to bring in state q'' all the non initial components of type $\mathcal{T}'$ that are in state q'. If $\mathcal{B}'_c(\langle \mathcal{T}', q''\rangle)$, in the reached configuration there are $j_{\langle \mathcal{T}', q''\rangle} + 1$ components of type $\mathcal{T}'$ in state q'' among those that do not belong to Z_0. In this case, a delete action is performed to reduce by 1 this quantity. The obtained configuration $\mathcal{C}_m$ has the required properties.
  The case when $\mathcal{B}'_c(\langle \mathcal{T}', q''\rangle) = \neg\mathcal{B}_c(\langle \mathcal{T}, q''\rangle)$ is analogous. □

Theorem 1. *The reconfiguration problem is PSpace.*

Proof. Given an abstract configuration checking if $\langle \mathcal{B}_i, \mathcal{B}_c \rangle \rightarrow \langle \mathcal{B}'_i, \mathcal{B}'_c \rangle$ can be done in polynomial space. This is due to the fact that $\langle \mathcal{B}_i, \mathcal{B}_c \rangle \rightarrow \langle \mathcal{B}'_i, \mathcal{B}'_c \rangle$ iff $\mathcal{B}_i$ and $\mathcal{B}_c$ differ from $\mathcal{B}'_i$ and $\mathcal{B}'_c$ for at most two $\langle \mathcal{T}, q \rangle$ pairs. Finding if an action can account for these differences can be done scanning the list of state change, new, and delete actions. As a consequence, the correctness of the Algorithm 1 (i.e., if it returns *True* then reconfiguration holds) follows from Lemma 3: it is sufficient to apply the statement of the lemma to the configuration obtained after having executed all possible binding actions on the initial configuration $\mathcal{C}_0$. Soundness (i.e., if reconfiguration holds then the algorithm returns *True*) follows from the following arguments. From Lemma 1 it is not restrictive to consider just the family of normalised deployment plans. If a solution to the reconfiguration problem exists, then for Lemma 2 an abstract deployment plan $\mathcal{B}_0 \rightarrow \cdots \rightarrow \mathcal{B}_n$ exists where $\mathcal{B}_n$ contains the target pair $\langle \mathcal{T}_t, q_t \rangle$. If n is smaller than the above upper bound K then the transitions in $\mathcal{B}_0 \rightarrow \cdots \rightarrow \mathcal{B}_n$ can be guessed by the algorithm so that it can return *True*. Consider now $n > K$. As discussed above, K is an upper bound to the number of different abstract configurations, hence there exist $i < j$ such that $\mathcal{B}_i = \mathcal{B}_j$. Consider now the abstract deployment plan $\mathcal{B}_0 \rightarrow \cdots \rightarrow \mathcal{B}_{i-1} \rightarrow \mathcal{B}_j \rightarrow \cdots \rightarrow \mathcal{B}_n$ which is strictly shorter than the previous one. If its length is smaller than K, this plan can be guessed by the algorithm, otherwise the same arguments can be applied to find another strictly shorter plan.

Abstract configurations $\langle \mathcal{B}_i, \mathcal{B}_c \rangle$ can be encoded with two tuples of size k. Since the $\mathcal{B}_i$ function considers only components in the initial configuration it can be represented with a tuple of values in the range $[0, |Z_0|]$: hence the possible values for $\mathcal{B}_i$ are $|Z_0|^k$. The function $\mathcal{B}_c$ instead can be encoded with a tuple of length k of booleans: hence its possible values are 2^k. The possible values for the counter variable are $|Z_0|^k * 2^k$. Assuming n the size of the input we have $|Z_0| \leq n, k \leq n$. Hence, such variables can be encoded in $\log(n^n + 2^n + n^n * 2^n) = O(\log(n^{2n})) = O(n \log(n))$. Hence the nondeterministic Algorithm 1 has polynomial space complexity. For this reason (and for Savitch's theorem [22]) we can conclude that the reconfiguration problem is in PSpace. □

B Proofs of Section 4

The proof of Theorem 2 requires some preliminary definitions and lemmas. We start by reporting the definition of our encoding of 1-safe Petri nets into Aeolus components.

Definition 13 (1-safe Petri net encoding). *Given a 1-safe Petri net* $N = (P, T, \boldsymbol{m_0})$ *its encoding is the set of component types* $\Gamma_N = \{\mathcal{T}_p \mid p \in P\} \cup \{\mathcal{T}_T\}$ *where* $\mathcal{T}_p$ *and* $\mathcal{T}_T$ *have been defined in Figure 3.*

Notice that the size of the component types Γ_N is polynomial w.r.t. the size of the 1-safe Petri net. This is due to the fact that place components have a constant number of states and ports while the component for the transitions has a number of states that grows linearly with respect to the number of places involved in the transitions.

We now introduce the notation $[\![\boldsymbol{m}]\!]$ to characterise configurations corresponding to the net marking $\boldsymbol{m}$. Intuitively, a configuration corresponds to a marking if it has just one component of type $\mathcal{T}_T$ and one component of type $\mathcal{T}_p$ for every place p. The component of type $\mathcal{T}_T$ must be in state q while the components $\mathcal{T}_p$ must be in on_p or off_p states according to the marking $\boldsymbol{m}$ considered. Moreover, the configuration could have as many components of type $\mathcal{T}_T$ or $\mathcal{T}_p$ in their initial state as wanted. In the following we use $\mathcal{C}^{\#}_{\langle \mathcal{T},q\rangle}$ to denote the number of components of $\mathcal{C}$ that have type $\mathcal{T}$ and are in state q.

Definition 14. *Let $N = (P, T, m_0)$ be a 1-safe Petri net and $\boldsymbol{m}$ one of its markings. We define:*

$$\begin{aligned}
[\![\boldsymbol{m}]\!] \;=\; \{\, \mathcal{C} \;\mid & \;\mathcal{C} \textit{ is a correct configuration having universe } \Gamma_N, \\
& \textstyle\sum_{q'\neq q_0} \mathcal{C}^{\#}_{\langle \mathcal{T}_T,q'\rangle} = 1,\; \mathcal{C}^{\#}_{\langle \mathcal{T}_T,q\rangle} = 1,\; \forall p \in P \,.\, \sum_{q'\neq q_0} \mathcal{C}^{\#}_{\langle \mathcal{T}_p,q'\rangle} = 1, \\
& \forall p \in \boldsymbol{m} \,.\, \mathcal{C}^{\#}_{\langle \mathcal{T}_p,on_p\rangle} = 1,\; \forall p \in P \setminus \boldsymbol{m} \,.\, \mathcal{C}^{\#}_{\langle \mathcal{T}_p,off_p\rangle} = 1 \,\}
\end{aligned}$$

We call *net step* a deployment plan that does not include new or delete actions and that starts with a state change of the component $\mathcal{T}_T$ in state q, and finishes with another state change of the same component that re-enters in such state q. Formally, it is a non empty sequence of reconfigurations $\mathcal{C}_1 \xrightarrow{\alpha_1} \mathcal{C}_2 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{m-1}} \mathcal{C}_m$ such that ${\mathcal{C}_1}^{\#}_{\langle \mathcal{T}_T,q\rangle} = {\mathcal{C}_m}^{\#}_{\langle \mathcal{T}_T,q\rangle} = 1$, while ${\mathcal{C}_i}^{\#}_{\langle \mathcal{T}_T,q\rangle} = 0$ for every $1 < i < m$, and α_j is neither a $new(z : \mathcal{T})$ nor a $del(z)$ action for every $1 \leq j \leq m-1$.

The proof of correspondence between a 1-safe Petri net N and its encoding Γ_N relies on two distinct lemmas, a first one about *completeness* of the simulation (i.e., each firing of a net transition can be mimicked by a deployment plan corresponding to a net step), and a second one about *soundness* (i.e., each net step corresponds to the firing of a net transition).

Lemma 4. *Let $N = (P, T, m_0)$ be a 1-safe Petri net, $\boldsymbol{m}$ one of its markings, and $\mathcal{C}$ a configuration in $[\![\boldsymbol{m}]\!]$. If $\boldsymbol{m} \mapsto \boldsymbol{m'}$ then there exists a net step from $\mathcal{C}$ to a configuration $\mathcal{C}' \in [\![\boldsymbol{m'}]\!]$.*

Proof. It is sufficient to observe that if $\boldsymbol{m} \mapsto \boldsymbol{m'}$ then there exists a transition $t \in T$ that, by consuming and producing tokens, transforms $\boldsymbol{m}$ in $\boldsymbol{m'}$. This transition can be selected in a deployment plan from $\mathcal{C}$ that starts by changing the state of $\mathcal{T}_T$ form q to q_t. Then the corresponding consumption and production phases can be executed and the component of type $\mathcal{T}_T$ can re-enter in state q. This deployment plan is a net step and the reached configuration is in $[\![\boldsymbol{m'}]\!]$. ☐

Lemma 5. *Let $N = (P, T, m_0)$ be a 1-safe Petri net, $\boldsymbol{m}$ one of its markings, and $\mathcal{C}$ a configuration in $[\![\boldsymbol{m}]\!]$ having a net step to $\mathcal{C}'$. Then, there exists $\boldsymbol{m'}$ s.t. $\mathcal{C}'$ is in $[\![\boldsymbol{m'}]\!]$ and $\boldsymbol{m} \mapsto \boldsymbol{m'}$.*

Proof. Let P be a net step from $\mathcal{C}$ to $\mathcal{C}'$ starting with a state change of the $\mathcal{T}_T$ component from state q to q_t, representing the selection of the transition t.

We first observe that, by definition of net step, the final configuration $\mathcal{C}'$ contains the $\mathcal{T}_T$ component in the q state. Moreover, since a net step does not involve delete actions there is exactly one component of type $\mathcal{T}_p$ per place p in a state different from the initial one. Finally, as the component of type $\mathcal{T}_T$ does not provide the ports a_p and b_p (for every $p \in P$) we have that these components must be either in *on* or *off* state. So there exists $\boldsymbol{m'}$ such that $\mathcal{C}' \in [\![\boldsymbol{m'}]\!]$.

Consider now that the net step from $\mathcal{C} \in [\![\boldsymbol{m}]\!]$ to $\mathcal{C}' \in [\![\boldsymbol{m'}]\!]$ includes the state changes, on the instance of component type $\mathcal{T}_T$, corresponding to the consumption and production phases for the selected transition t. The execution of the consumption phase guarantees that ${}^\bullet t \leq \boldsymbol{m}$, thus t can fire in $\boldsymbol{m}$. Moreover, the consumption (resp. production) phase guarantees that a token component is moved from *on* to *off* (resp. *off* to *on*) iff it belongs to ${}^\bullet t$ (resp. $t^\bullet$). Hence, we can conclude that $\boldsymbol{m} \mapsto \boldsymbol{m'}$ as effect of the firing of t. □

Notice that besides the net step from $\mathcal{C}$ to $\mathcal{C}'$ considered in the above Lemma, there are deployment plans starting from $\mathcal{C}$ that do not correspond to net steps. Some of them have anyway the same effect of a net step because they reach a configuration in a set $[\![\boldsymbol{m''}]\!]$ s.t. $\boldsymbol{m} \mapsto \boldsymbol{m''}$. Other plans do not reach such a configuration because, e.g., they are composed of an infinite sequence of creations and deletions of components or could select a non enabled transition t. The presence of these plans is irrelevant, as far as the reconfiguration problem is concerned, because they cannot deploy the considered target component.

We can now conclude that given the 1-safe Petri net $N = \langle P, T, \boldsymbol{m_0} \rangle$ and a target marking $\boldsymbol{m_t}$, we can consider the set of component types Γ_N and a component type $\mathcal{T}_A$ having, beside an initial state, a state q_a reachable from it that requires the port t, the port on_p for all places $p \in \boldsymbol{m_t}$ and port off_p for all places $p \notin \boldsymbol{m_t}$. Lemmas 4 and 5 (and the observation after Lemma 5) guarantee that $\boldsymbol{m_t}$ can be reached in N *iff* the component type-state pair $\langle \mathcal{T}_A, q_a \rangle$ can be deployed with the universe of component types $\Gamma_N \cup \{\mathcal{T}_A\}$ starting from any of the configurations in $[\![\boldsymbol{m_0}]\!]$. PSpace-hardness of reachability in 1-safe Petri nets [3] implies PSpace-hardness of the reconfiguration problem.

Theorem 2. *The reconfiguration problem is PSpace-hard.*

Proof. Consider a 1-safe Petri net $N = \langle P, T, \boldsymbol{m_0} \rangle$ and a target marking $\boldsymbol{m_t}$. The problem of checking whether $\boldsymbol{m_t}$ can be reached in N from a marking $\boldsymbol{m_0}$ is PSpace-hard [3]. Consider the set of component types Γ_N and a component type $\mathcal{T}_A$ having, beside an initial state, a state q_a reachable from the initial state that requires the port t, a port on_p for all places $p \in \boldsymbol{m_t}$ and port off_p for all places $p \notin \boldsymbol{m_t}$. We have already observed that the size of Γ_N is polynomial w.r.t. the size of the 1-safe net N. The size of the component type $\mathcal{T}_A$ is also polynomial w.r.t. the size of the 1-safe net N since it has just 2 states, 2 state transitions, and $|P| + 1$ ports.

We complete the proof by showing that a marking $\boldsymbol{m_t}$ in N can be reached *iff* the component type-state pair $\langle \mathcal{T}_A, q_a \rangle$ is achievable with the universe of component types $\Gamma_N \cup \{\mathcal{T}_A\}$ starting from a configuration in $[\![\boldsymbol{m_0}]\!]$.

The “only if” part follows from Lemma 4, that guarantees the existence of a deployment plan reaching a configuration $\mathcal{C} \in [\![\boldsymbol{m_t}]\!]$. If $\mathcal{C}^{\#}_{\langle \mathcal{T}_A, q_a \rangle} = 1$ then the thesis is proven. Otherwise, from the configuration $\mathcal{C}$ it is possible to create a new component z of type $\mathcal{T}_A$, bind each of its ports to the component of type $\mathcal{T}_T$ in state q and to the components of type $\mathcal{T}_p$ in *on* or *off* state, and perform a state change of z into q_a.

The proof of the “if” part proceeds as follows. Starting from a configuration $\mathcal{C}_0 \in [\![\boldsymbol{m_0}]\!]$, the achievability of the pair $\langle \mathcal{T}_A, q_a \rangle$ guarantees the existence of a deployment plan $\mathsf{P} = \mathcal{C}_0 \xrightarrow{\alpha_1} \mathcal{C}_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} \mathcal{C}_n$ where the configuration $\mathcal{C}_n$ has a component of type $\mathcal{T}_A$ in state q_a. We can construct another deployment plan $\mathsf{P}' = \mathcal{C}_0 \xrightarrow{\alpha'_1} \cdots \xrightarrow{\alpha'_m} \mathcal{C}'_l$ by considering only the actions in $\alpha_1, \cdots, \alpha_n$ that involves components of type $\mathcal{T}_p$ or $\mathcal{T}_T$ not in their initial state. This new deployment plan does not contain new and delete actions, and it turns out to be a sequence of net steps such that $\mathcal{C}'_l \in [\![\boldsymbol{m_t}]\!]$. By Lemma 5 there exists a sequence of net transitions $\boldsymbol{m_0} \mapsto \cdots \mapsto \boldsymbol{m_t}$ thus proving the thesis. $\square$