ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Architecture Diagrams — A Graphical Language for Architecture Style Specification

EPFL IC IIF RiSD Technical Report
EPFL-REPORT-215210
http://infoscience.epfl.ch/record/215210

Anastasia Mavridou, Eduard Baranov, Simon Bliudze, Joseph Sifakis

May 12, 2016

Abstract: Architecture styles characterise families of architectures sharing common characteristics. We have recently proposed configuration logics for architecture style specification. In this technical report, we study a graphical notation to enhance readability and easiness of expression. We study simple architecture diagrams and a more expressive extension, interval architecture diagrams. For each type of diagrams, we present its semantics, a set of necessary and sufficient consistency conditions and a method that allows to characterise compositionally the specified architectures. We provide several examples illustrating the application of the results. We also present a polynomial-time algorithm for checking that a given architecture conforms to the architecture style specified by a diagram.

```
@TechReport{MBBS16-ModellingArchs-TR,
  author = {Anastasia Mavridou, Eduard Baranov, Simon Bliudze, Joseph Sifakis},
  title = {Architecture Diagrams - A Graphical Language for Architecture Style Specification},
  institution = {EPFL IC IIF RiSD},
  year = 2016,
  number = {EPFL-REPORT-215210},
  note = {Available at: \texttt{http://infoscience.epfl.ch/record/215210}}
}
```

Contents

1 Introduction

Software architectures [25, 27] describe the high-level structure of a system in terms of components and component interactions. They depict generic coordination principles between types of components and can be considered as generic operators that take as argument a set of components to be coordinated and return a composite component that satisfies by construction a given characteristic property [2].

Many languages have been proposed for architecture description, such as architecture description languages (e.g. [21, 11]), coordination languages (e.g. [24, 1]) and configuration languages (e.g. [28, 16]). All these works rely on the distinction between behaviour of individual components and their coordination in the overall system organization. Informally, architectures are characterized by the structure of the interactions between a set of typed components. The structure is usually specified as a relation, e.g. connectors between component ports.

Architecture styles characterise not a single architecture but a family of architectures sharing common characteristics, such as the type of the involved components and the topology induced by their coordination structure. Simple examples of architecture styles are Pipeline, Ring, Master/Slave, Pipe and Filter. For instance, Master/Slave architectures integrate two types of components masters and slaves such that each slave can interact only with one master. Figure 1 depicts four Master/Slave architectures involving two master components M_1, M_2 and two slave components S_1, S_2. Their communication ports are respectively p_1, p_2 and q_1, q_2. A Master/Slave architecture for two masters and two slaves can be represented as one among the following configurations, i.e. sets of connectors: $\{p_1q_1, p_2q_2\}$, $\{p_1q_2, p_2q_1\}$, $\{p_1q_1, p_1q_2\}$, $\{p_2q_1, p_2q_2\}$. A term p_iq_j represents a connector between ports p_i and q_j. The four architectures are depicted in Figure 1. The Master/Slave architecture style denotes all the Master/Slave architectures for arbitrary numbers of masters and slaves.

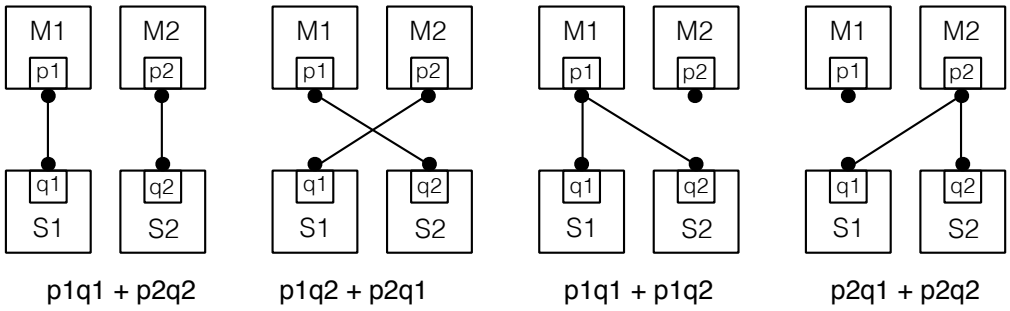

Figure 1: Master/Slave architectures.

We have recently proposed configuration logics [20] for the description of architecture styles. These are powerset extensions of interaction logics [3] used to describe architectures. In addition to the operators of the extended logic, they have logical operators on sets of architectures. We have studied higher-order configuration logics and shown that they are a powerful tool for architecture style specification. Nonetheless, their richness in operators and concepts may make their use challenging.

In this paper we explore a different avenue to architecture style specification based on *architecture diagrams*. Architecture diagrams describe the structure of a system by showing the system's component types and their attributes for coordination, as well as relationships among component types. Our notation allows the specification of generic coordination mechanisms based on the concept of *connector*.

Architecture diagrams were mainly developed for architecture style specification in BIP [2],

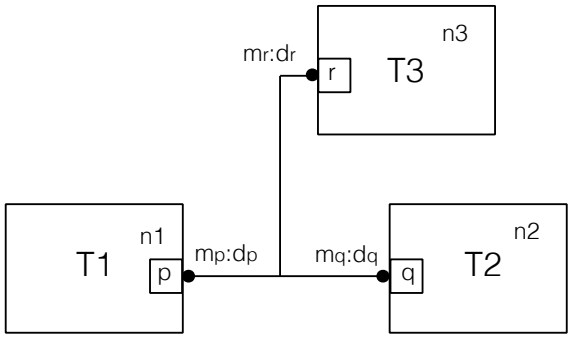

Figure 2: An architecture diagram.

where connectors are defined as n-ary synchronizations among component ports and do not carry any additional behaviour. Nevertheless, our approach can be extended for architecture style specification in other languages by explicitly associating the required behaviour to connectors.

An architecture diagram consists of a set of *component types*, a *cardinality function* and a set of *connector motifs*. Component types are characterised by sets of *generic ports*. The cardinality function associates each component type with its *cardinality*, i.e. number of instances. Figure 2 shows an architecture diagram consisting of three component types T_1, T_2 and T_3 with n_1, n_2 and n_3 instances and generic ports p, q and r, respectively. Instantiated components have *port instances* p_i, q_j, r_k for i, j, k belonging to the intervals $[1, n_1]$, $[1, n_2]$, $[1, n_3]$, respectively.

Connector motifs are non-empty sets of generic ports that must interact. Each generic port p in the connector motif has two constraints represented as a pair $m : d$. Multiplicity m is the number of port instances p_i that are involved in the connectors. Degree d specifies the number of connectors in which each port instance is involved. A connector motif defines a set of configurations, where a configuration is a set of connectors. The architecture diagram of Figure 2 has a single connector motif involving generic ports p, q and r.

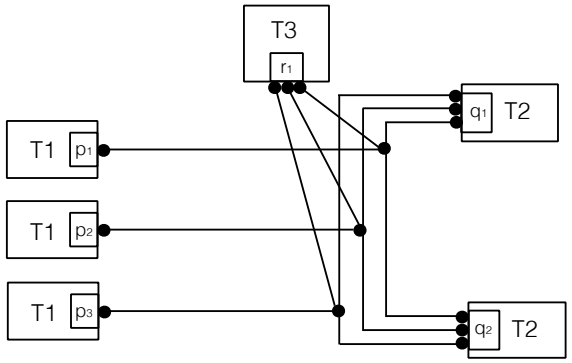

Figure 3: Architecture obtained from diagram of Figure 2.

Figure 3 shows the unique architecture obtained from the diagram of Figure 2 by taking $n_1 = 3$, $m_p = 1$, $d_p = 1$; $n_2 = 2$, $m_q = 2$, $d_q = 3$; $n_3 = 1$, $m_r = 1$, $d_r = 3$. This is the result of composition of constraints for generic ports p, q and r as depicted in Figure 4. For port p, we have three instances and as both the multiplicity and the degree are equal to 1, each port p_i has a single connector lead. For port q, we have two instances and as the multiplicity is 2, we have connectors involving q_1 and q_2 and their total number is equal to 3 to meet the degree constraint. Finally, for

port r, we have a single instance r_1 that has three connector leads to satisfy the degree constraint.

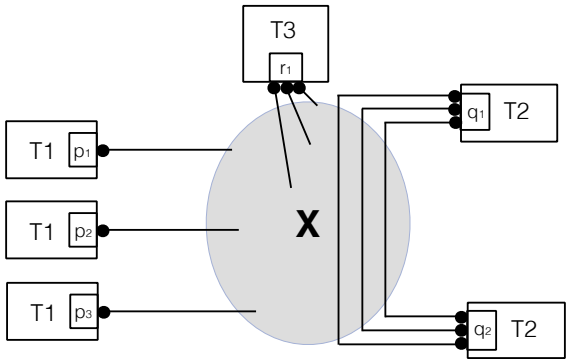

Figure 4: Composition for the diagram of Figure 2.

The semantics of an architecture diagram consisting of a set of connector motifs $\{\Gamma_i\}_{i\in[1..k]}$ is defined as follows. The meaning of each connector motif Γ_i as a set of configurations $\{\gamma_{i,j}\}_{j\in J_i}$. The architecture diagram specifies all the architectures characterised by configurations of connectors of the form: $\gamma_{1,j_1} \cup \cdots \cup \gamma_{n,j_n}$, where the indices $j_i \in J_i$. In other words, the configuration of an architecture conforming to the diagram is obtained by taking the union of all sub-configurations corresponding to each connector motif.

We study a method that allows to characterise compositionally the set of configurations specified by a given connector motif if *consistency conditions* are met. It involves a two-step process. The first step consists in characterising configuration sets meeting the coordination constraints for each generic port p of the connector motif. In the second step, the configuration of the connector motif is obtained by fusing one by one connectors obtained from step one so that the multiplicities and the degrees of the ports are preserved.

We study two types of architecture diagrams: *simple architecture diagrams* and *interval architecture diagrams*. In the former the cardinality, multiplicity and degree constraints are positive integers, while in the latter they can be also intervals. We show that interval architecture diagrams are strictly more expressive than simple architecture diagrams. For each type of diagrams we present 1) its syntax and semantics; 2) a set of consistency conditions; 3) a method that allows to characterise compositionally all the configurations of a connector motif; 4) multiple examples of architecture style specification. Finally, we present a polynomial-time algorithm for checking that a given diagram conforms to the architecture style specified by a diagram.

The report is structured as follows. Section 2 presents simple architecture diagrams. Section 3 presents interval architecture diagrams. Section 4 presents a polynomial-time algorithm for checking conformance of diagrams. Section 5 discusses related work. Section 6 concludes the report by summarising the results and discussing possible directions for future work.

2 Simple Architecture Diagrams

2.1 Syntax and Semantics

We focus on the specification of generic coordination mechanisms based on the concept of connector. Therefore, the nature and the operational semantics of components are irrelevant. We consider that a component interface is defined by its set of ports, which are used for interaction with other components. Thus, a *component type* T has a set of *generic ports* $T.P$.

A *simple architecture diagram* $\langle \mathcal{T}, n, \mathcal{C} \rangle$ consists of:

- a set of *component types* $\mathcal{T} = \{T_1, \ldots, T_k\}$;
- an associated *cardinality* function $n : \mathcal{T} \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers (to simplify the notation, we will abbreviate $n(T_i)$ to n_i);
- a set of *connector motifs* $\mathcal{C} = \{\Gamma_1, \ldots, \Gamma_l\}$ of the form $\Gamma = (a, \{m_p : d_p\}_{p \in a})$, where $\emptyset \neq a \subset \bigcup_{i=1}^{k} T_i.P$ is a generic connector and $m_p, d_p \in \mathbb{N}$ (with $m_p > 0$) are the *multiplicity* and *degree* associated to generic port $p \in a$.

Figure 5 shows the graphical representation of an architecture diagram with a single connector motif. It defines different architecture styles, for different values of the multiplicity, degree and cardinality parameters.

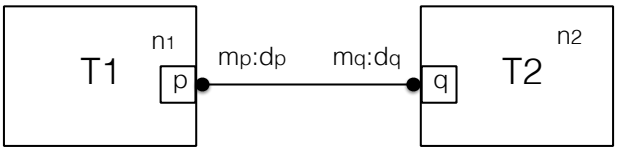

Figure 5: A simple architecture diagram.

An *architecture* is a pair $\langle \mathcal{B}, \gamma \rangle$, where $\mathcal{B}$ is a set of components and γ is a *configuration*, i.e. a set of connectors among the ports of components in $\mathcal{B}$. We define a connector as a set of ports that must interact. For a component $B \in \mathcal{B}$ and a component type T, we say that B *is of type* T if the ports of B are in a bijective correspondence with the generic ports in T. Let $B_1, \ldots, B_n$ be all the components of type T in $\mathcal{B}$. For a generic port $p \in T.P$, we denote the corresponding port instances by $p_1, \ldots, p_n$ and its associated cardinality by $n_p = n(T)$.

Semantics. An architecture $\langle \mathcal{B}, \gamma \rangle$ *conforms* to a diagram $\langle \mathcal{T}, n, \mathcal{C} \rangle$ if, for each $i \in [1, k]$, the number of components of type T_i in $\mathcal{B}$ is equal to n_i and γ can be partitioned into disjoint sets $\gamma_1, \ldots, \gamma_l$, such that, for each connector motif $\Gamma_i = (a, \{m_p : d_p\}_{p \in a}) \in \mathcal{C}$ and each $p \in a$,

1. there are exactly m_p instances of p in each connector in γ_i and
2. each instance of p is involved in exactly d_p connectors in γ_i.

The meaning of a simple architecture diagram is a set of all architectures that conform to it.

We assume that, for any two connector motifs $\Gamma_i = (a, \{m_p^i : d_p^i\}_{p \in a})$ (for $i = 1, 2$) with the same set of generic ports a, there exists $p \in a$, such that $m_p^1 \neq m_p^2$. Two connector motifs with the same set of generic ports and multiplicities $\Gamma_i = (a, \{m_p : d_p^i\}_{p \in a})$ (for $i = 1, 2$) can be replaced by a single connector motif $\Gamma = (a, \{m_p : d_p^1 + d_p^2\}_{p \in a})$, which imposes a weaker constraint. To be more specific, by this assumption, we lose the ability to guarantee that a set of connectors γ corresponding to a connector motif $(a, \{m_p : d_p\}_{p \in a})$ not only satisfies the multiplicity and degree constraints but also can be split into several disjoint subsets $\gamma_1, \ldots, \gamma_n$ such that for each γ_i for $i \in [1, n]$ for each $p \in a$ and for any $p_j, p_k \in p$, $|\{b \in \gamma_i | p_j \in b\}| = |\{b \in \gamma_i | p_k \in b\}|$, i.e. the degree of all port instances of a generic port are equal.

We consider that the aforementioned assumption does not have significant impact on the expressiveness of the formalism. On the contrary, it greatly simplifies semantics and analysis. In particular, it ensures that for any configuration γ there exists at most one partition into disjoint sets $\gamma_1, \ldots, \gamma_l$, which correspond to different connector motifs. In other words, a connector cannot correspond to two different connector motifs. This greatly simplifies function `VerifyMultiplicity` in Subsection 4. Furthermore, it also simplifies the consistency conditions presented in Subsection 2.2. Notice that this assumption allows connector motifs with the same set of generic ports but different multiplicities.

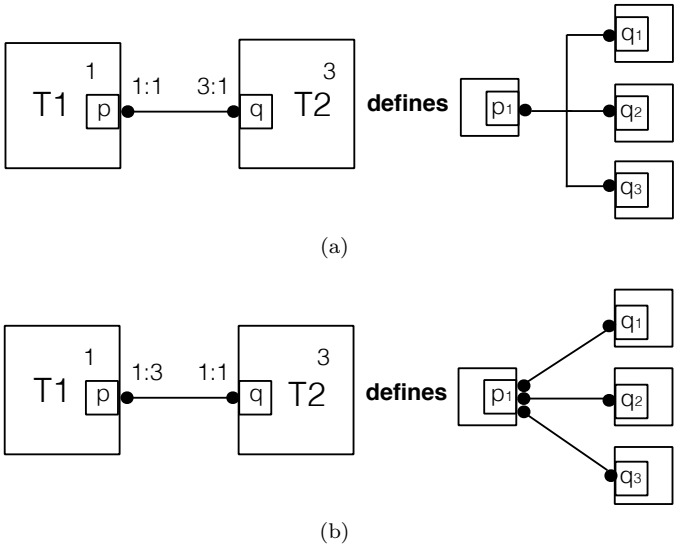

Figure 6: Two examples of simple architecture diagrams.

Multiplicity constrains the number of instances of the generic port belonging to a connector involved in a motif, whereas degree constrains the number of connectors attached to any instance of the generic port. Consider the two diagrams shown in Figures 6(a) and 6(b). They have the same set of component types and cardinalities. Nevertheless, their multiplicities and degrees differ, resulting in different architectures. Architectures conforming to the two diagrams are also shown in Figures 6(a) and 6(b).

In Figure 6(a), the multiplicity of the generic port p is 1 and the multiplicity of the generic port q is 3, thus, any connector must involve one instance of p and three instances of q. Since there are only three instances of q, any connector must include all of them. The degrees of both generic ports are 1, so each port is involved in exactly one connector. Thus, the diagram defines a single architecture with one connector among the four ports.

In Figure 6(b) the multiplicity of both generic ports p and q are 1. Thus, any connector must involve one instance of p and one instance of q. Since the degree of q is 1 and there are three instances of q we need three connectors, each involving a distinct instance of q. Thus, the architecture diagram defines a single architecture with three binary connectors.

2.2 Consistency Conditions

Notice that there exist diagrams that do not define any architecture such as the diagram shown in Figure 7. Since the multiplicity is 1 for both generic ports p and q, a configuration in a corresponding architecture must include only binary connectors involving one instance of p and one instance of q. Additionally, since the degree of both p and q is 1, each port instance must be involved in exactly one connector. However, the cardinalities impose that there be three connectors attached to the instances of p, but only two connectors attached to the instances of q. Both requirements cannot be satisfied simultaneously and therefore, no architecture can conform to this diagram.

Consider a connector motif $\Gamma = (a, \{m_p : d_p\}_{p \in a})$ in a diagram $\langle \mathcal{T}, n, \mathcal{C} \rangle$ and a generic port $p \in a$, such that $p \in T.P$, for some $T \in \mathcal{T}$. We denote $s_p = n_p \cdot d_p / m_p$ the *matching factor of* p.

A *regular configuration of* p is a multiset of connectors, such that 1) each connector involves

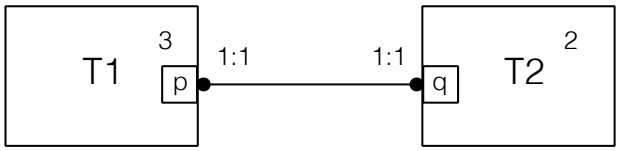

Figure 7: An inconsistent diagram.

m_p instances of p and no other ports and 2) each instance of the port p is involved in exactly d_p connectors. Notice the difference between a configuration and a regular configuration of p: the former defines a set of connectors, while the latter defines a multiset of sub-connectors involving only instances of the generic port p. Considering the diagram in Figure 2 and the architecture in Figure 3 the only regular configuration of r is the multiset $\{r_1,\ r_1,\ r_1\}$. The three copies of the singleton sub-connector r_1 are then fused with sub-connectors $p_i q_1 q_2$ $(i = 1, 2, 3)$, resulting in a configuration with three distinct connectors.

Lemma 2.1. *Each regular configuration of a port p has exactly s_p connectors.*

Proof. 1) we have s_p connectors and each connector is a set of m_p port instances. Thus the sum of connectors sizes is $s_p \cdot m_p$; 2) Connectors consist of ports and each port instance is involved in d_p connectors. The total number of ports in connectors is $n_p \cdot d_p$ where n_p is the number of port instances. Thus $s_p \cdot m_p = n_p \cdot d_p$ or $s_p = n_p \cdot d_p / m_p$. □

Notice that, for the diagram of Figure 7, we have $s_p = 3$, while $s_q = 2$. To form connectors, each sub-connector from a regular configuration of p has to be fused with exactly one sub-connector from a regular configuration of q, and vice-versa. Since, by the above lemma, the sizes of such regular configurations are different, there is no architecture conforming to this diagram.

Proposition 2.2 provides the necessary and sufficient conditions for a simple architecture diagram to be consistent, i.e. to have at least one conforming architecture. The multiplicity of a generic port must be less than or equal to the number of component instances that contain this port. The matching factors of all ports participating in the same connector motif must be equal integers. Finally, there must be enough distinct sub-connectors to build a configuration. Notice that because of the assumption that we made in Subsection 2.1 this condition can be applied independently to each connector motif. Since, by the semantics of diagrams, connector motifs correspond to disjoint sets of connectors, these conditions are applied separately to each connector motif.

Proposition 2.2. *A simple architecture diagram has a conforming architecture iff, for each connector motif $\Gamma = (a, \{m_p : d_p\}_{p \in a})$ and each $p \in a$, we have*

1. $m_p \leq n_p$,
2. $\forall q \in a,\ s_p = s_q \in \mathbb{N}$,
3. $s_p \leq \prod_{q \in a} \binom{n_q}{m_q}$.

Proof. This proposition is a special case of Proposition 3.5. □

2.3 Synthesis of Configurations

The synthesis procedure for each connector motif consists of the following two steps: 1) we find regular configurations for each generic port satisfying the connector motif constraints; 2) we fuse these regular configurations generating global configurations specified by the connector motif.

Table 1: Vector representation of regular configurations.

$d_p = 1$	$[100001], [010010], [001100]$.
$d_p = 2$	$[110011], [101101], [011110],$
	$[200002], [020020], [002200]$.
$d_p = 3$	$[111111], [210012], [201102], [120021], [021120],$
	$[012210], [102201], [300003], [030030], [003300]$.

2.3.1 Regular Configurations of a Generic Port

We start with an example illustrating the steps of the synthesis procedure for a port p.

Example 2.3. Consider a port p with $n_p = 4$ and $m_p = 2$. There are 6 connectors of multiplicity 2: p_1p_2, p_1p_3, p_1p_4, p_2p_3, p_2p_4, p_3p_4. They correspond to the set of edges of a complete graph with vertices p_1, p_2, p_3, p_4. The regular configurations of p for $d_p = 1, 2, 3$, where each edge appears at most once (i.e. sets of connectors) are shown in Figure 8.

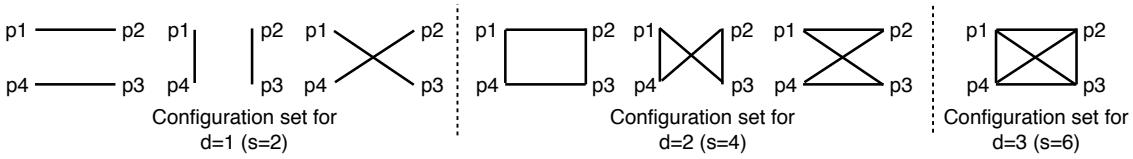

Figure 8: Regular configurations of p with $n_p = 4$, $m_p = 2$.

We provide below an equational characterisation of all the regular configurations (multisets) of a given generic port p with given n_p, m_p, and d_p. For the n_p port instances, $p_1, \ldots, p_n$ we have a set $\{a_i\}_{i\in[1,w]}$ of different connectors, where $w = \binom{n_p}{m_p}$, to which we associate a column vector of non-negative integer variables $X = [x_1, \ldots, x_w]^T$.

Consider the Example 2.3 and variables $x_1, \ldots, x_6$ representing the number of occurrences in a regular configuration of the connectors p_1p_2, p_1p_3, p_1p_4, p_2p_3, p_2p_4, p_3p_4, respectively. All the regular configurations (i.e. multisets of connectors), for $d_p = 1, 2, 3$, represented as vectors of the form $[x_1, \ldots, x_6]$ are listed in Table 1. Notice that vectors for $d_p > 1$ can be obtained as linear combinations of the vectors describing configuration sets for $d_p = 1$.

Then, for the port p we define an $n_p \times w$ incidence matrix $G = [g_{i,j}]_{n_p\times w}$ with $g_{i,j} = 1$ if $p_i \in a_j$ and $g_{i,j} = 0$ otherwise. The following equation holds: $GX = D$, where $D = [d_p, \ldots, d_p]$ (d_p repeated n_p times). Any non-negative integer solution of this equation defines a regular configuration of p. For Example 2.3, the equations are:

$$\begin{cases} x_1 + x_2 + x_3 = d, \\ x_1 + x_4 + x_5 = d, \\ x_2 + x_4 + x_6 = d, \\ x_3 + x_5 + x_6 = d\,, \end{cases} \quad \text{i.e.} \quad \begin{cases} x_1 + x_2 + x_3 = d, \\ x_3 = x_4, \\ x_2 = x_5, \\ x_1 = x_6\,. \end{cases} \tag{1}$$

Notice that the vectors of Table 1 are solutions of (1).

2.3.2 Configurations of a Connector Motif

Let $\Gamma = (a, \{m_p : d_p\}_{p\in a})$ be a connector motif such that all generic ports of $a = \{p^1, \ldots, p^v\}$ have the same integer matching factor s. For each $p^j \in a$, let $\gamma^j = \{a_i^j\}_{i\in[1,s]}$ be a regular configuration of p^j. For arbitrary permutations π_j of $[1, s]$, a set $\{a_i^1 \cup \bigcup_{j=2}^{v} a^j_{\pi_j(i)}\}_{i\in[1,s]}$ is a configuration specified by the connector motif.

In order to provide an equational characterisation of the connector motif, we consider, for each $j \in [1, v]$, a corresponding solution vector X^j of equations $G^jX^j = D^j$ characterising the regular configurations of p^j (cf. Section 2.3.1). Denote w^j the dimension of the vector X^j.

In order to characterise the configurations of connectors conforming to Γ, we consider, for each configuration, the v-dimensional matrix $E = [e_{i_1,\ldots,i_v}]_{w^1\times\cdots\times w^v}$ of 0-1 variables, such that $e_{i_1,\ldots,i_v} = 1$ if the connector $a^1_{i_1} \cup \cdots \cup a^v_{i_v}$ belongs to the configuration and 0 otherwise. By definition, the sum of all elements in E is equal to s. Moreover, the following equations hold:

$$\begin{cases} x_i^1 = \Sigma_{i_2,i_3\ldots,i_v}\ e_{i,i_2,\ldots,i_v}\,, & \text{for } i \in [1, w^1], \\ x_i^2 = \Sigma_{i_1,i_3,\ldots,i_v}\ e_{i_1,i,\ldots,i_v}\,, & \text{for } i \in [1, w^2], \\ \quad\vdots & \\ x_i^v = \Sigma_{i_1,i_2,\ldots,i_{v-1}}\ e_{i_1,\ldots,i_{v-1},i}\,, & \text{for } i \in [1, w^v]. \end{cases} \tag{2}$$

For instance, for a fixed $i \in [1, w^1]$, all $e_{i,i_2,\ldots,i_v}$ describe all connectors that contain a_i^1. The regular configuration γ^1 is characterised by X^1, enforcing that a_i^1 belongs to x_i^1 connectors. The system of linear equations (2), combined with the systems of linear equations $G^jX^j = D^j$, for $j \in [1, v]$, fully characterise the configurations of Γ. They can be used to synthesise architectures from architecture diagrams.

Example 2.4. Consider a diagram $\big(\{T_1, T_2\}, n, \{\Gamma\}\big)$, where $T_1 = \{p\}$, $T_2 = \{q\}$, $n(T_1) = n(T_2) = 4$ and $\Gamma = (pq, \{(m_p : d_p, m_q : d_q)\})$ with $m_p = 2$, $m_q = 3$. The corresponding equations $G_pX = D_p$, $G_qY = D_q$ can be rewritten as

$$\begin{cases} x_1 + x_2 + x_3 = d_p, \\ x_3 = x_4,\ x_2 = x_5,\ x_1 = x_6, \end{cases} \quad \text{and} \quad \begin{cases} 3y_1 = d_q, \\ y_1 = y_2 = y_3 = y_4. \end{cases} \tag{3}$$

Together with the constraints $x_i = \Sigma_j e_{i,j}$ and $y_j = \Sigma_i e_{i,j}$, for $E = [e_{i,j}]_{6\times4}$, equations (3) completely characterise all the configurations conforming to Γ.

The same methodology can be used to synthesise configurations with additional constraints. To impose that some specific connectors must be included, whereas other specific connectors must be excluded from the configurations, the corresponding variables in the matrix E are given fixed values: 1 (resp. 0) if the connector must be included (resp. excluded) from the configurations. The rest of the synthesis procedure remains the same.

Example 2.5. Figure 9 shows the architecture diagram from Example 2.4, with $d_p = 2$ and $d_q = 3$. We want to synthesise the configurations of this diagram with the following additional constraints: connectors $p_1p_2q_1q_2q_3$ and $p_1p_3q_2q_3q_4$ must be included, whereas connector $p_2p_4q_1q_2q_4$ must be excluded from the synthesised configurations.

First, we compute the vectors X and Y that represent the regular configurations of generic ports p and q, respectively. Variables $x_1, \ldots, x_6$ represent the number of occurrences in a configuration of the connectors p_1p_2, p_1p_3, p_1p_4, p_2p_3, p_2p_4, p_3p_4, respectively. Variables $y_1, \ldots, y_4$ represent the number of occurrences in a configuration of the connectors $q_1q_2q_3$, $q_1q_2q_4$, $q_1q_3q_4$, $q_2q_3q_4$, respectively.

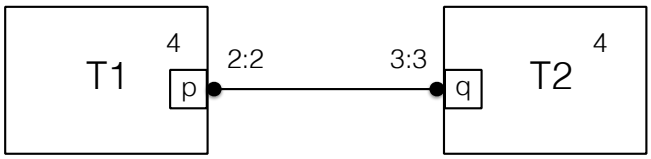

Figure 9: Architecture diagram of Example 2.5.

Vector X can take one of the following values for $d_p = 2$: [110011], [101101], [011110], [200002], [020020] or [002200] (Example 2.3). Regular configurations of q are characterised by the equations $3y_1 = d$ and $y_1 = y_2 = y_3 = y_4$ (Example 2.4). For $d = 3$ there is a single solution $Y = [1111]$.

We now consider the matrix E, where we fix $e_{1,1} = e_{2,4} = 1$ and $e_{5,2} = 0$ to impose the additional synthesis constraints:

$$E = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & e_{1,2} & e_{1,3} & e_{1,4} \\ e_{2,1} & e_{2,2} & e_{2,3} & 1 \\ e_{3,1} & e_{3,2} & e_{3,3} & e_{3,4} \\ e_{4,1} & e_{4,2} & e_{4,3} & e_{4,4} \\ e_{5,1} & 0 & e_{5,3} & e_{5,4} \\ e_{6,1} & e_{6,2} & e_{6,3} & e_{6,4} \end{pmatrix} \end{array}$$

Since, for all $i \in [1, 6]$, we have $x_i = \Sigma_j\, e_{i,j}$, we observe that $x_1, x_2 \geq 1$. The only valuation of X that satisfies this constraint is [110011]; as mentioned above, the only possible valuation of Y is [1111].

The sum of rows 3 and 4 of E is 0, so all their elements must be 0s. The sum of rows 1 and 2 as well as the sum of columns 1 and 4 is 1. Since there exists already an element with value 1, all other elements in these rows and columns must be 0s. This gives us the following multidimensional matrix:

$$E = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & e_{5,3} & 0 \\ 0 & e_{6,2} & e_{6,3} & 0 \end{pmatrix} \end{array}$$

The sums of the rest rows and columns give us the correct values of the other three elements. The complete solution is the following:

$$E = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array},$$

which corresponds to the configuration $\{p_1p_2q_1q_2q_3,\ p_1p_3q_2q_3q_4,\ p_2p_4q_1q_3q_4,\ p_3p_4q_1q_2q_4\}$.

2.4 Architecture Style Specification Examples

Example 2.6. The Star architecture style consists of a single center component of type $T_1 = \{p\}$ and n_2 components of type $T_2 = \{q\}$. The central component is connected to every other component by a binary connector and there are no other connectors. The diagram in Figure 10 graphically describes the Star architecture style.

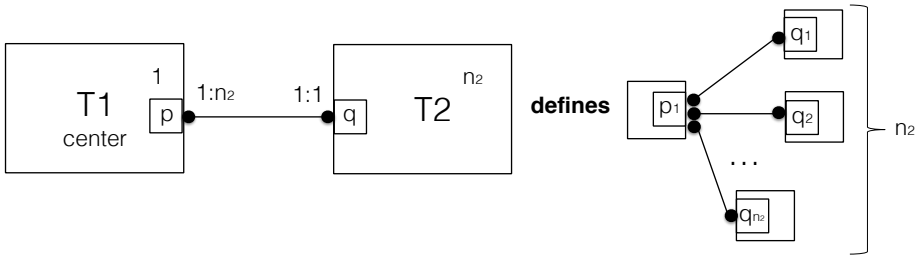

Figure 10: Star architecture style.

Example 2.7. We now consider the multi-star extension of the Star architecture style, with n center components of type T_1, each connected to d components of type T_2 by binary connectors. As in Example 2.6, there are no other connectors. The diagram of Figure 11 graphically describes this architecture style.

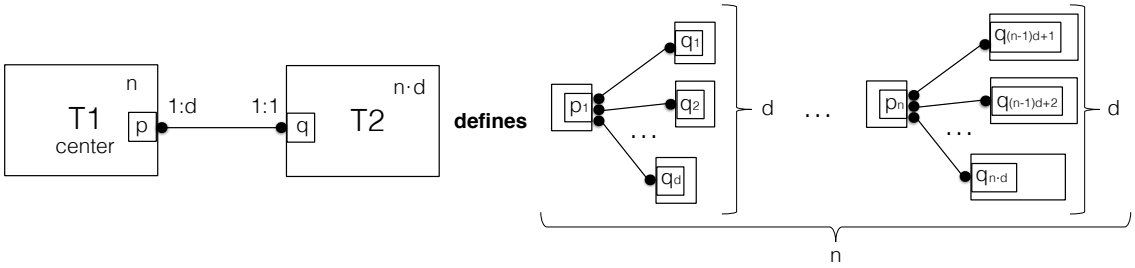

Figure 11: Multi-star architecture style.

3 Interval Architecture Diagrams

To further enhance the expressiveness of diagrams we introduce interval architecture diagrams where the cardinalities, multiplicities and degrees parameters can be intervals. With simple architecture diagrams we cannot express properties such as *"component instances of type T are optional"*. For instance, let us consider the example of Figure 1 that shows four Master/Slave architectures involving two masters and two slaves. In this example, one of the masters might be optional, i.e. it might not interact with any slaves. As illustrated in Figure 1, in the first and second architectures each master interacts with one slave, however, in the third and fourth architectures a single master interacts with both slaves while the other master does not interact with any slaves. In other words, the degree of the generic port m varies from 0 to 2 and cannot be represented by an integer.

3.1 Syntax and Semantics

An *interval architecture diagram* $\langle \mathcal{T}, n, \mathcal{C} \rangle$ consists of:

- a set of *component types* $\mathcal{T} = \{T_1, \ldots, T_k\}$;
- a *cardinality* function $n : \mathcal{T} \to \mathbb{N}^2$, associating, to each $T_i \in \mathcal{T}$, an interval $n(T_i) = [n_i^l, n_i^u] \subset \mathbb{N}$ (thus, $n_i^l \leq n_i^u$);
- a set of *connector motifs* $\mathcal{C} = \{\Gamma_1, \ldots, \Gamma_l\}$ of the form $\Gamma = \left(a, \{ty[m_p^l, m_p^u] : ty[d_p^l, d_p^u]\}_{p \in a}\right)$, where $\emptyset \neq a \subset \bigcup_{i=1}^{k} T_i.P$ is a generic connector and $ty[m_p^l, m_p^u], ty[d_p^l, d_p^u]$, with $[m_p^l, m_p^u]$, $[d_p^l, d_p^u] \subset \mathbb{N}$ non-empty intervals and $ty \in \{mc, sc\}$ (mc means "multiple choice", whereas sc means "single choice"), are, respectively, *multiplicity* and *degree* constraints associated to $p \in a$.

Semantics. An architecture $\langle \mathcal{B}, \gamma \rangle$ *conforms* to an interval architecture diagram $\langle \mathcal{T}, n, \mathcal{C} \rangle$ if, for each $i \in [1, k]$, the number of components of type T_i in $\mathcal{B}$ lies in $[n_i^l, n_i^u]$ and γ can be partitioned into disjoint sets $\gamma_1, \ldots, \gamma_l$, such that for each connector motif $\Gamma_i = \left(a, \{ty[m_p^l, m_p^u] : ty[d_p^l, d_p^u]\}_{p \in a}\right) \in \mathcal{C}$ and each $p \in a$:

1. there are $m_p \in [m_p^l, m_p^u]$ instances of p in each connector in γ_i; in case of a single choice interval the number of instances of p is equal in all connectors in γ_i;
2. each instance of p is involved in $d_p \in [d_p^l, d_p^u]$ connectors in γ_i; in case of a single choice interval, the number of connectors involving an instance of p is the same for all instances of p.

The meaning of an interval architecture diagram is a set of all architectures that conform to it.

In other words, each generic port p has an associated pair of intervals defining its multiplicity and degree. The interval attributes specify whether these constraints are uniformly applied or not. We write $sc[x, y]$ (single choice) to mean that the same multiplicity or degree is applied to each port instance of p. We write $mc[x, y]$ (multiple choice) to mean that different multiplicities or degrees can be applied to different port instances of p, provided they lie in the interval.

We assume that, for any two connector motifs $\Gamma_i = (a, \{ty[m_p^l, m_p^u]_i : ty[d_p^l, d_p^u]_i\}_{p \in a})$ (for $i = 1, 2$) with the same set of generic ports a, there exists $p \in a$ such that $[m_p^l, m_p^u]_1 \cap [m_p^l, m_p^u]_2 = \emptyset$. Similarly to simple architecture diagrams, without significant impact on the expressiveness of the formalism, this assumption greatly simplifies semantics and analysis. In particular, it ensures that for any configuration γ there exists at most one partition into disjoint sets $\gamma_1, \ldots, \gamma_l$, which correspond to different connector motifs, which simplifies function `VerifyMultiplicity` in Section 4. Furthermore, it simplifies the consistency conditions in Proposition 3.5 that can now be applied independently to each connector motif.

Example 3.1. The diagram in Figure 12 defines the set of architectures shown in Figure 1. Notice that the degree of the generic port p is the multiple choice interval $[0, 2]$, since one master component may be connected to two slaves, while the other master may have no connections.

Proposition 3.2. *Interval architecture diagrams are strictly more expressive than simple architecture diagrams.*

Proof. Any parameter x of a simple architecture diagram can be represented as $sc[x, x]$ in interval architecture diagrams. Thus, any simple architecture diagram can be represented as an interval architecture diagram. The diagram of Example 3.1 cannot be expressed with simple architecture diagrams, proving that interval architecture diagrams are strictly more expressive. □

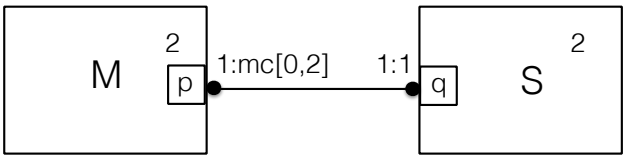

Figure 12: Architecture diagram of Figure 1.

3.2 Consistency Conditions

Similarly to simple architecture diagrams, there are interval diagrams that do not define any architectures. Proposition 3.5 provides the necessary and sufficient conditions for the consistency of interval architecture diagrams. A connector cannot contain more port instances than there exist in the system. Thus, the lower bound of multiplicity should not exceed the maximal number of instances of the associated component type. For all generic ports of a connector motif, there should exist a common matching factor that does not exceed the maximum number of different connectors between these ports. These conditions are a generalisation of Proposition 2.2. Lemmas 3.3 and 3.4 are auxiliary for proving Proposition 3.5.

Lemma 3.3. *Consider a set of generic ports P and a set of s connectors over these ports. Assume that connector $k \in [1, s]$ contains $m_{k,p}$ port instances of $p \in P$ and a port instance $p_j \in p$ is an element of exactly $d_{p,j}$ connectors. The following equality holds $\forall p \in P$, $\sum_{k=1}^{s} m_{k,p} = \sum_{p_j \in p} d_{p,j}$.*

Proof. Consider a generic port $p \in P$. Consider a bipartite graph $G = (U, V, E)$ defined by two disjoint sets of vertices: set of vertices U where each vertex corresponds to one port instance, and a set of vertices V where each vertex corresponds to one connector. The graph G has an edge between vertices $u \in U$ and $v \in V$ if the port associated to u is an element of the connector associated to v. A vertex $v_k \in V$ associated with connector k is adjacent to exactly $m_{k,p}$ vertices associated with ports p of the connector k. Therefore, the number of edges between U and V is equal to $\sum_{k=1}^{s} m_{k,p}$. Furthermore, the degree of a vertex $u_j \in U$ is equal to $d_{p,j}$ because p_j is an element of $d_{p,j}$ connectors. Therefore, the number of edges between U and V must also be equal to $\sum_{p_j \in p} d_{p,j}$. Combining this with our previous observation, we obtain that $\sum_{k=1}^{s} m_{k,p} = \sum_{p_j \in p} d_{p,j}$. The same reasoning can be applied to any generic port $p \in P$ giving the same equality. □

Lemma 3.4. *Let P be a set of generic ports with two associated parameters: n_p representing a number of port instances $p \in P$ and $[d_p^l, d_p^u]$ for $d_p^l \in \mathbb{N}, d_p^u \in \mathbb{N},$, $d_p^l \leq d_p^u$ representing the desired degree interval. Consider a set of s distinct connectors A over P, such that for a port $p \in P$, a connector $a \in A$ contains $m_{a,p}$ instances of p, where $m_{a,p} \leq n_p$, and $n_p \cdot d_p^l \leq \sum_{a \in A} m_{a,p} \leq n_p \cdot d_p^u$. Then it is possible to construct a set of s distinct connectors A' such that a connector a contains $m_{a,p}$ instances of p with the degree of an instance p_j being equal to $d_{p,j} \in [d_p^l, d_p^u]$.*

Proof. Let $d_{p,i}$ be a degree of the port p_i in A, i.e. $d_{p,i} = |\{a \in A | p_i \in a\}|$. Let us define a function $f : 2^A \to \mathbb{N}$ such that $f(A) = \Sigma_{p \in P} \Sigma_{p_i \in p} \min_{d_p \in [d_p^l, d_p^u]} |d_{p,i} - d_p|$. Function $f(A)$ achieves its minimal value $f(A) = 0$ if and only if the degree $d_{p,i} \in [d_p^l, d_p^u]$ for all ports. That is, if $f(A) = 0$, we construct A' by the assignment $A' = A$.

Suppose now that we have A for which $f(A) \neq 0$. Since $f(A) \neq 0$, this means that there is at least one port p_i such that $d_{p,i} < d_p^l$ or $d_{p,i} > d_p^u$. Without loss of generality we can assume the first case (the other case is symmetric). From Lemma 3.3, we know that $\sum_{k=1}^{s} m_{k,p} = \sum_{p_j \in p} d_{p,j}$. Thus, for at least one port $p_j \in p$ holds $d_{p,j} > d_p^l$.

Now, observe that for the two ports p_i and p_j, $d_{p,j} > d_{p,i}$. This means that we can redefine at least one connector by replacing port p_i with port p_j without having duplicated connectors

(otherwise, by the pigeonhole principle port p_j would already be an element of two identical connectors). Consider a new set of s connectors A_{new} obtained by applying the port replacement procedure. Its value function is at most $f(A_{new}) \leq f(A) - 1 < f(A)$. Since initial value $f(A)$ is bounded, the consecutive application of the port replacement procedure eventually leads to set A_{new} for which $f(A_{new}) = 0$. Therefore, it is possible to construct set A'. □

To simplify the presentation of Proposition 3.5 we use the following notion of choice function. Let $\mathcal{I}_T$ and $\mathcal{I}$ be the sets of, respectively, typed intervals and intervals, as in the definition of interval diagrams above. A function $g : \mathcal{I}_T \to \mathcal{I}$ is a *choice function* if it satisfies the following constraints:

$$g(ty[x,y]) = \begin{cases} [x,y], & \text{if } ty = mc, \\ [z,z], & \text{for some } z \in [x,y], \text{ if } ty = sc. \end{cases}$$

Proposition 3.5. *An interval architecture diagram $\langle \mathcal{T}, n, \mathcal{C} \rangle$ is consistent iff, for each $T \in \mathcal{T}$, there exists a cardinality $n_i \in [n_i^l, n_i^u]$ and, for each connector motif $(a, \{M_p : D_p\}_{p \in a}) \in \mathcal{C}$ and each $p \in a$, there exist choice functions g_p^m, g_p^d, such that, for $[m_p^l, m_p^u] = g_p^m(M_p)$ and $[d_p^l, d_p^u] = g_p^d(D_p)$ hold:*

1. *$m_p^l \leq n_p$, for all $p \in a$, (where $n_p = n_i$ for $p \in T.P$),*
2. *$(S \cap U \cap \mathbb{N}) \neq \emptyset$, where*

   (a) $S = \bigcap_{p \in a} s_p$ *with* $s_p = \begin{cases} [\frac{n_p \cdot d_p^l}{m_p^u}, \frac{n_p \cdot d_p^u}{m_p^l}], & \text{if } m_p^l > 0, \\ [\frac{n_p \cdot d_p^l}{m_p^u}, \infty), & \text{if } m_p^l = 0, \end{cases}$

   (b) $U = [1, \prod_{p \in a} \sum_{m \in [m_p^l, m_p^u]} \binom{n_p}{m}]$.

Proof. Necessity →: Consider an architecture conforming to the diagram. Consider values $n_T \in [n_T^l, n_T^u]$ for each $T \in \mathcal{T}$ equal to the number of components of the corresponding type in the architecture. Consider a connector motif $\Gamma = (a, \{M_p : D_p\}_{p \in a}) \in \mathcal{C}$ and consider functions g_p^m, g_p^d for each generic port $p \in a$ consistent with the architecture, i.e. if the multiplicity (degree) of p in the sub-configuration corresponding to the connector motif Γ in the architecture is equal to v and the multiplicity (degree) interval has type sc then the corresponding g_p^m (g_p^d) returns $[v, v]$.

Condition 1 is trivially obtained - $n_p < m_p^l$ cannot occur as the multiplicity of ports cannot be greater than the number of component instances. In order to show condition 2 we apply Lemma 3.3, $\forall p \in a$, $\sum_{k=1}^{s} m_{k,p} = \sum_{p_j \in p} d_{p,j}$, where s is the number of connectors in the architecture corresponding to Γ. The lower bound on the left hand side is $s \cdot m_p^l$, while the upper bound on the right hand side is $n_p \cdot d_p^u$: these two bounds give us $s \leq \frac{n_p \cdot d_p^u}{m_p^l}$ (if $m_p^l = 0$, $s \to \infty$). By inspecting the upper bound of the left hand side and the lower bound of the right hand side, we obtain $s \geq \frac{n_p \cdot d_p^l}{m_p^u}$. Thus, $s \in S$. For the set U, notice that $\prod_{p \in a} \sum_{m \in [m_p^l, m_p^u]} \binom{n_p}{m}$ is equal to the number of different ways one could connect ports in a, so that port $p \in a$ has n_p instances and connector k contains $m_{k,p} \in [m_p^l, m_p^u]$ ports $p_i \in p$. Therefore, $s \in U$, otherwise, by the pigeonhole principle there would exist duplicated connectors. Thus, $s \in S$ and $s \in U$ and $s \in \mathbb{N}$ so their intersection is not empty. This reasoning can be applied to any connector motif, proving the necessity of the consistency conditions.

Sufficiency ←: We prove this part by construction. Consider values n_T and functions g for which all conditions are satisfied. Consider a set of behaviours, such that each type $T \in \mathcal{T}$ has n_T instances. In order to construct an architecture we need only a set of connectors. We construct

sets for each connector motif independently, taking their union in the final step. Consider a connector motif $\Gamma = (a, \{M_p : D_p\}_{p \in a}) \in \mathcal{C}$. Suppose that there are no degree constraints. As in the first part of the proof, we know that condition 2 implies that the number of connectors is bounded by $\prod_{p \in a} \sum_{m \in [m_p^l, m_p^u]} \binom{n_p}{m}$, where $m_p^l \leq n_p$, which is satisfied by condition 1. Since $\prod_{p \in a} \sum_{m \in [m_p^l, m_p^u]} \binom{n_p}{m}$ is the number of different ways one could connect ports in a, so that generic port $p \in a$ has n_p instances and connector k contains $m_{k,p} \in [m_p^l, m_p^u]$ port instances of generic port p, it follows that it is always possible to select s distinct connectors (distinct by the set of port instances they contain), where a port instance $p_i \in p$ is allowed to have a degree $d_{p,i} \notin [d_p^l, d_p^u]$. Now, consider one such set A of s connectors. Since condition 2 is satisfied, we know that $n_p \cdot d_p^l \leq \sum_{k \in A} m_{k,p} \leq n_p \cdot d_p^u$ for all $p \in a$, so we can apply the result of Lemma 3.4. More precisely, we can iterate over ports in a (in arbitrary order), and balance the degrees of port instances $p_i \in p$, achieving the degree $d_{p,i} \in [d_p^l, d_p^u]$. Since by Lemma 3.4 each iteration preserves the distinctness of connections, once the entire iterative procedure finishes, we obtain a set of s distinct connectors for which each port instance has the degree that takes values in $[d_p^l, d_p^u]$, and this holds for all $p \in a$. Considering such sets for each connector motif and taking their union, we obtain an architecture that conforms to the diagram. □

3.3 Synthesis of Configurations

The equational characterisation in Section 2.3 can be generalised, using systems of inequalities with some additional variables, to interval architecture diagrams. Below, we show how to characterise the configurations induced by n instances of a generic port p with the associated degree interval $ty[d_p^l, d_p^u]$.

For a given multiplicity m, let $X = [x_1, \ldots, x_w]^T$ be the column vector of integer variables, corresponding to the set $\{a_i\}_{i \in [1,w]}$ (with $w = \binom{n}{m}$) of connectors of multiplicity m, involving port instances $p_1, \ldots, p_n$. Let G be the incidence matrix $G = [g_{i,j}]_{n \times w}$ with $g_{i,j} = 1$ if $p_i \in a_j$ and $g_{i,j} = 0$ otherwise. The configurations induced by the n instances of p are characterised by the equation $GX = D$, where $D = [d_1, \ldots, d_n]^T$ and the additional (in)equalities:

$$\begin{array}{ll} d_1 = \cdots = d_n = d \text{ and } d_p^l \leq d \leq d_p^u, & \text{for } ty = sc, \\ d_p^l \leq d_1 \leq d_p^u, \ldots, d_p^l \leq d_n \leq d_p^u, & \text{for } ty = mc. \end{array} \tag{4}$$

Example 3.6. As in Example 2.3, consider a generic port p and $n = 4$, $m = 2$. For the degree interval $sc[1, 3]$, the corresponding constraints are $1 \leq d \leq 3$, $x_1 + x_2 + x_3 = d$, $x_4 = x_3$, $x_5 = x_2$, $x_6 = x_1$.

For the degree interval $mc[1, 3]$ the corresponding constraints are $1 \leq d_i \leq 3$, for $i \in [1, 4]$, $x_1 + x_2 + x_3 = d_1$, $x_1 + x_4 + x_5 = d_2$, $x_2 + x_4 + x_6 = d_3$, $x_3 + x_5 + x_6 = d_4$. By solving this system we get:

$$\begin{cases} 0 \leq x_i \text{ for } i \in [1..6], \\ x_6 \leq 3, \\ x_5 \leq 3 - x_6, \\ x_4 \leq 3 - x_6, \\ x_4 \leq 3 - x_5, \\ 1 - x_5 - x_6 \leq x_3 \leq 3 - x_5 - x_6, \\ 1 - x_4 - x_6 \leq x_2 \leq 3 - x_4 - x_6, \\ x_2 \leq 3 - x_3, \\ 1 - x_4 - x_5 \leq x_1 \leq 3 - x_4 - x_5, \\ 1 - x_2 - x_3 \leq x_1 \leq 3 - x_2 - x_3. \end{cases}$$

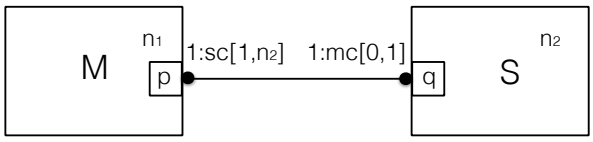

Figure 13: Master/Slave architecture style.

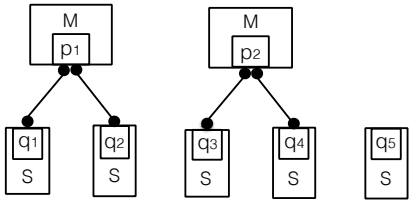

Figure 14: A Master/Slave architecture.

Suppose that the multiplicity of p in the motif is given by an interval $ty[m_p^l, m_p^u]$. Contrary to the degree, multiplicity does not appear explicitly as a variable in the constraints. Instead, it influences the number and nature of elements in both the matrix G and vector X.

Therefore, for single choice (i.e. $ty = sc$), the configurations induced by n instances of p are characterised by the disjunction of the instantiations of the system of equalities combining $G_m X_m = D$ with (4), for $m \in [m_p^l, m_p^u]$.

For multiple choice (i.e. $ty = mc$), all the configurations are characterised by the system combining (4) with

$$\sum_{m \in [m_p^l, m_p^u]} (G_m X_m) = D\,.$$

Notice that the above modifications to accommodate for interval-defined multiplicity are orthogonal to those in (4), accommodating for interval-defined degree. Similarly to the single-choice case for multiplicity, for interval-defined cardinality, the configurations are characterised by taking the disjunction of the characterisations for all values $n \in [n_p^l, n_p^u]$.

Based on the above characterisation for the configurations of one generic port, global configurations can be characterised by systems of linear constraints in the same manner as for simple architecture diagrams.

3.4 Architecture Style Specification Examples

Example 3.7. The diagram of Figure 13 describes a particular Master/Slave architecture style. We require that each slave interact with at most one master and that each master be connected to the same number of slaves.

Multiplicities of both generic ports p and q are equal to 1, allowing only binary connectors between a master and a slave. The single choice degree of generic port p ensures that all port instances are connected to the same number of connectors which is a number in $[1, n_2]$. The multiple choice degree of generic port q ensures that all port instances are connected to at most one master. A conforming architecture for $n_1 = 2$ and $n_2 = 5$ is shown in Figure 14.

Example 3.8. The diagram in the top of Figure 15 describes the Repository architecture style involving a single instance of a component of type R and an arbitrary number of data-accessor components of type A. We require that any data-accessor component must be connected to the repository. In the bottom of Figure 15, we show conforming architectures for for 3 data-accessors.

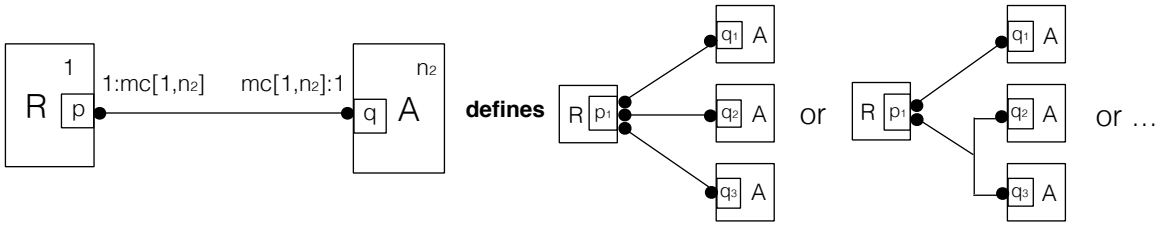

Figure 15: Repository architecture style.

Example 3.9. The Pipes and Filters architecture style [7] involves two types of components, P and F, each having two ports *in* and *out*. Each input (resp. output) of a filter is connected to an output (resp. input) of a single pipe. The output of any pipe can be connected to at most one filter. Figure 16 graphically describes the Pipes and Filters architecture style. A conforming architecture for $n_1 = 3$ and $n_2 = 4$ is shown in Figure 17.

Figure 16: Pipes and Filters architecture style.

Example 3.10. The Peer-to-Peer architecture style [7] involves one component type P with generic ports *req* (request) and *pro* (provide). Figure 18 graphically describes the Peer-to-Peer architecture style. Peers request and provide services by using binary connectors between their *req* and *pro* ports. A conforming architecture for $n_1 = 4$ is also shown in Figure 18.

Example 3.11. The Map-Reduce architecture style [6] allows processing large datasets, such as those found in search engines and social networking sites. Figure 19 graphically describes the Map-Reduce architecture style. A conforming architecture for $n_1 = 3$ and $n_2 = 2$ is shown in Figure 20.

A large dataset is split into smaller datasets and stored in the global filesystem (GFS). The $Master$ is responsible for coordinating and distributing the smaller datasets from the GFS to each of the map worker components (MW). The port *in* of each MW is connected to the $Mcontrol$ and *read* ports of the $Master$ and the GFS, respectively. Each MW processes the datasets and writes the result to its dedicated local filesystem (LFS) through a binary connector between their *out* and *write* ports. The connector is binary since no MW is allowed to read the output of another MW. Each reduce worker (RW) reads the results from multiple LFS as instructed by

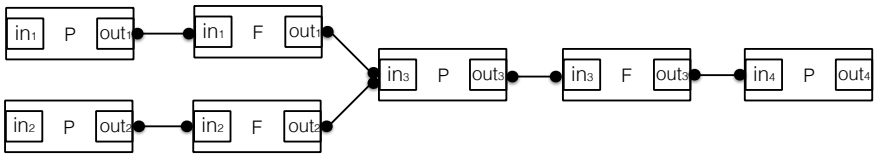

Figure 17: A Pipes and Filters architecture.

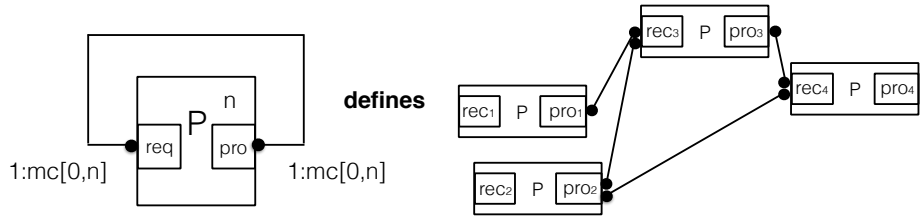

Figure 18: Peer-to-Peer architecture style.

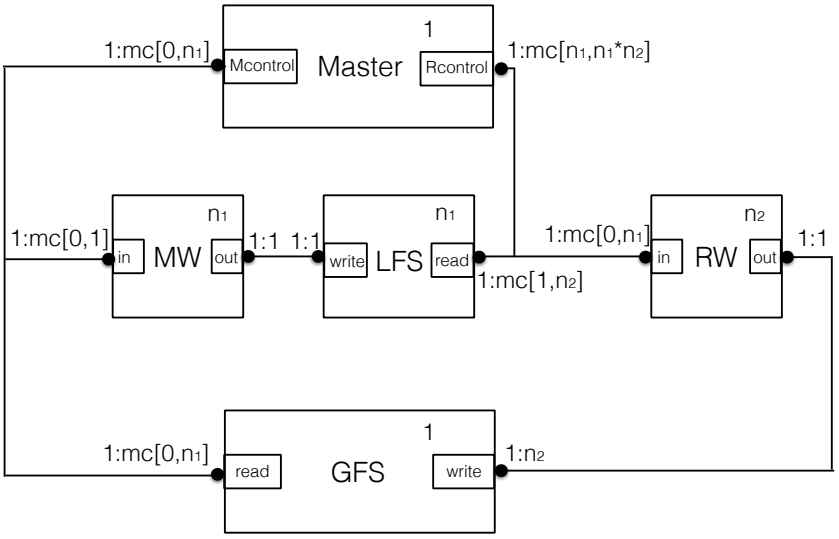

Figure 19: Map-Reduce architecture style.

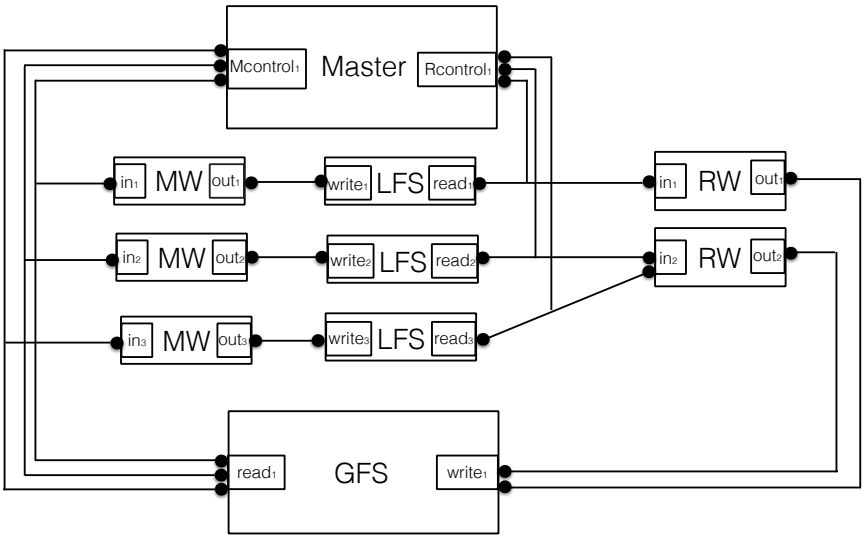

Figure 20: A Map-Reduce architecture.

the $Master$ component. To this end, the in port of each RW is connected to the $Rcontrol$ and $read$ ports of the $Master$ and some LFS, respectively. Then, each RW combines the results to produce a final result written back to the GFS through a binary connector between their out and $write$ ports. No RW is allowed to read the output of another RW.

4 Checking Conformance of Diagrams

Algorithm 1 can be used to check whether an architecture $\langle B, \gamma \rangle$ conforms to an interval architecture diagram $\langle \mathcal{T}, n, \mathcal{C} \rangle$. This algorithm has polynomial-time complexity.

Algorithm 1 checks the validity of the following three statements: 1) the number of components of each type T is equal to $n(T)$; 2) there exists a partition of γ into $\gamma_1, \ldots, \gamma_l$ such that each γ_i corresponds to a different connector-motif $\Gamma_i \in \mathcal{C}$ of the diagram and satisfies its multiplicity constraints; 3) for each connector motif Γ_i and its corresponding γ_i, the number of times each port instance participates in γ_i satisfies the degree requirements. The aforementioned statements correspond to functions VerifyCardinality, VerifyMultiplicity and VerifyDegree, respectively.

We use two auxiliary functions. Function $generic(p)$ takes a port instance and returns a corresponding generic port. Function $typeof(B)$ returns the component type of component B. Operation $map[key]++$ increases the $value$ associated with the key by one if the key is already in the map, otherwise it adds a new key to the map with $value$ 1.

Algorithm 1: VerifyArchitecture

```
Data: Architecture ⟨B, γ⟩, diagram ⟨T, n, C⟩
Result: Returns true if the architecture satisfies the diagram ⟨T, n, C⟩. Otherwise, it
        returns false.
begin
    if not VerifyCardinality(B, ⟨T, n, C⟩) then
        return false;
    /* Splits connectors between connector motifs according to multiplicities constraints.*/
    S_γ ⟵ VerifyMultiplicity(γ, C);
    if S_γ = ∅ then
        return false;
    /* Verifies degree constraints for all generic ports of all connector motifs. */
    for Γ ∈ C do
        if VerifyDegree(S_γ[Γ], Γ) ≠ true then
            return false;
    return true;
```

5 Related Work

A plethora of approaches exist for architecture specification. Patterns [5, 10] are commonly used for specifying architectures in practical applications. The specification of architectures is usually done in a graphical way using general purpose graphical tools. Such specifications are easy to produce but the meaning of the design may not be clear since the graphical conventions lack formal semantics and thus, are not amenable to formal analysis.

Function VerifyCardinality($\mathcal{B}$, $\langle \mathcal{T}, n, \mathcal{C} \rangle$)

Data: Set of components B, diagram $\langle \mathcal{T}, n, \mathcal{C} \rangle$
Result: Returns *true* if the number of components of each type in B lies in the corresponding cardinality interval specified in the diagram. Otherwise, it returns $false$.

```
begin
    /* Creates a map with key: component type, value: number of instances */
    countTypes ⟵ {};
    for B_i ∈ B do
        countTypes[typeof(B_i)] + +;
    for T_i ∈ T do
        if countTypes[T_i] ∉ n(T_i) then
            return false;
    return true;
```

Researchers have developed a number of Architecture Description Languages (ADLs) for architecture specification [21, 29, 23]. Nevertheless, according to [19], architectural languages used in practice mostly originate from industrial development instead of academic research. Practitioners insist on using UML rather than ADLs. UML and some of the ADLs lack formal semantics. ADLs that have formal semantics require the use of formal languages. According to [19], ADLs that rely on formal languages have proven to be difficult for practitioners to master. Keeping this in mind, we have created architecture diagrams that combine the benefits of graphical languages and rigorous formal semantics. By relying on the minimal set of notions, we emphasize the conceptual clarity of our approach.

Architecture diagrams were developed to accommodate architecture specification in BIP [2], wherein connectors are n-ary relations among ports and do not carry any additional behaviour. This strict separation of computation from coordination allows reasoning about the coordination constraints structurally and independently from the behaviour of coordinating components. However, our approach can be extended to describe architecture styles in other coordination languages by explicitly associating the required behaviour to connector motifs. In particular, this can be applied to specify connector patterns in Reo [1], by associating multiplicity and degree to source and sink nodes of connectors. The main difficulty is to correctly instantiate the behaviour depending on the number of ends in the connector instance.

Alloy [13] has been used for architecture style specification, in the ACME [14] and Darwin [8] ADLs. The connectivity primitives in [14, 8] are binary predicates and cannot tightly characterize coordination structures involving multiparty interaction. To specify an n-ary interaction, these approaches require an additional entity connected by n binary links with the interacting ports. Since the behaviour of such entities is not part of the architecture style, it is impossible to distinguish, e.g. between an n-ary synchronisation and a sequence of n binary ones.

Architecture diagrams consist of component types and connector motifs, respectively comparable to UML components and associations [12, 22]. One important difference between connector motifs used in our architecture diagrams and UML associations is that the latter cannot specify interactions that involve two or more instances of the same component type [22]. In UML, the term "multiplicity" is used to define both 1) the number of instances of a UML component and 2) the number of UML links connected to a UML component. In architecture diagrams, we call these, respectively, "cardinality" and "degree". We use the term "multiplicity" to denote the number

Function VerifyMultiplicity(γ, $\mathcal{C}$)

Data: Configuration γ, Set of connector motifs $\mathcal{C}$
Result: Returns a partition $\mathcal{P}_\gamma$ of γ such that each part corresponds to one $\Gamma \in \mathcal{C}$. Connectors in each part satisfy multiplicity constraint of the corresponding connector motif. If no partition exists, returns $\emptyset$.

begin
  /* Creates a map for the partition with key: connector motif and value: sub-configuration.*/
  $partition \longleftarrow \{\}$;
  for $\Gamma \in \mathcal{C}$ **do**
    $partition[\Gamma] \longleftarrow \emptyset$;
  /* Creates a map for the single choice intervals with key: generic port of a connector motif and value: chosen value.*/
  $scValues \longleftarrow \{\}$;

  for $k \in \gamma$ **do**
    /* Creates a map with key: generic port and value: number of instances of the generic port in the connector. */
    $portsCount \longleftarrow \{\}$;
    for $p_i \in k$ **do**
      $portsCount[generic(p_i)] + +$;
    $x \longleftarrow false$;
    /* Tries to find a connector motif such that connector satisfies its constraints.*/
    for $\Gamma = (a, \{ty[m_p^l, m_p^u] : d_p\}_{p \in a}) \in \mathcal{C}$ **do**
      if $a = keys(portsCount)$ **then**
        $y \longleftarrow true$;
        /* Checks the multiplicity intervals.*/
        for $p \in a$ **do**
          if $portsCount[p] \notin [m_p^l, m_p^u]$ **then**
            $y \longleftarrow false$;
            break;
          /* Additional check in case of the single choice interval.*/
          if $ty[m_p^l, m_p^u] = sc[m_p^l, m_p^u]$ **then**
            if $hasKey(scValues, \langle \Gamma, p \rangle)$ && $scValues[\langle \Gamma, p \rangle] \neq portsCount[p]$ **then**
              $y \longleftarrow false$;
              break;
            else
              $scValues[\langle \Gamma, p \rangle] \longleftarrow portsCount[p]$;
        if y **then**
          $partition[\Gamma] \longleftarrow partition[\Gamma] \cup k$;
          $x \longleftarrow true$;
          break;
    /* A connector does not satisfy constraints of any connector motif. */
    if $x = false$ **then**
      return $\emptyset$;
  return partition;

Function VerifyDegree(γ_i, Γ)

Data: Configuration γ_i, Connector motif $\Gamma = (a, \{m_p : ty[d_p^l, d_p^u]\}_{p \in a})$
Result: Returns *true* if the degree requirements are satisfied. Otherwise returns false.
begin
  /* Creates a map with key: port and value: number of connectors it appears in.*/
  $degrees \longleftarrow \{\}$;

  for $k \in \gamma_i$ **do**
    for $p_i \in k$ **do**
      $degrees[p_i] + +$;

  /* Creates a map for the single choice intervals with key: generic port and value: chosen value.*/
  $scValues \longleftarrow \{\}$;

  for $p_i \in keys(degrees)$ **do**
    $p \longleftarrow generic(p_i)$;
    if $degrees[p_i] \notin [d_p^l, d_p^u]$ **then**
      return $false$;
    /* Additional check in case of the single choice interval.*/
    if $ty[d_p^l, d_p^u] = sc[d_p^l, d_p^u]$ **then**
      if $hasKey(scValues, p)$ && $scValues[p] \neq degrees[p_i]$ **then**
        return $false$;
      else
        $scValues[p] \longleftarrow degrees[p_i]$;
  return $true$;

of components of the same class that can be connected by the same connector. The distinction between multiplicity and degree is key for allowing n-ary connectors involving several instances of the same component type.

A large body of literature, originating in [9, 18], studies the use of graph grammars and transformations [26] to define software architectures. Although this work focuses mainly on dynamic reconfiguration of architectures, e.g. [4, 15, 17], graph grammars can be used to define architecture styles: a style admits all the configurations that can be derived by its defining grammar. A limitation of this approach already outlined in [18] is the difficulty of understanding the architecture style defined by a grammar. Our approach allows intuitive specification of architecture styles since all constraints appear directly in the architecture diagram for which we provide denotational semantics.

6 Conclusion and Future Work

We study architecture diagrams, a graphical language rooted in well-defined semantics for the description of architecture styles. We study two classes of diagrams. Simple architecture diagrams express uniform degree and multiplicity constraints. They are easy to interpret and use but they have limited expressive power. Interval architecture diagrams are strictly more expressive. They cannot be modelled as the union of simple architecture diagrams, because they allow heterogeneity

of multiplicity and degree. Architecture diagrams provide powerful and flexible means for graphical specification of architectures with n-ary connectors. Using architecture diagrams instead of purely logic-based specifications confers the advantages of graphical formalisms.

In an ongoing project partially financed by the European Space Agency, we are already using architecture diagrams to describe some of the architectures in the case studies of the project. We are currently working on extending the current notation with arithmetic constraints and implementing the synthesis procedure described in this technical report with the JaCoP[1] constraint solver. In the future, we plan to extend connector motifs with data flow information. We also plan to study the expressive power of architecture diagrams and compare it with that of configuration logics.

References

[1] Farhad Arbab. Reo: A channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[2] Paul Attie, Eduard Baranov, Simon Bliudze, Mohamad Jaber, and Joseph Sifakis. A general framework for architecture composability. *Formal Aspects of Computing*, pages 1–25, 2015.

[3] Simon Bliudze and Joseph Sifakis. The algebra of connectors — Structuring interaction in BIP. *IEEE Transactions on Computers*, 57(10):1315–1330, 2008.

[4] Roberto Bruni, Alberto Lluch-Lafuente, Ugo Montanari, and Emilio Tuosto. Style-based architectural reconfigurations. *Bulletin of the EATCS*, 94:161–180, 2008.

[5] Robert Daigneau. *Service design patterns: Fundamental design solutions for SOAP/WSDL and restful Web Services*. Addison-Wesley, 2011.

[6] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[7] David Garlan and Mary Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, pages 1–39. World Scientific Publishing Company, 1993.

[8] Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems*, pages 33–38. ACM, 2002.

[9] Dan Hirsch, Paola Inverardi, and Ugo Montanari. Modeling software architectures and styles with graph grammars and constraint solving. In Patrick Donohoe, editor, *Software Architecture*, volume 12 of *IFIP*, pages 127–143. Springer, 1999.

[10] Gregor Hohpe and Bobby Woolf. *Enterprise integration patterns: designing, building, and deploying messaging solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[11] ISO/IEC/IEEE 42010. *Systems and software engineering — Architecture description*, 2011.

[12] James Ivers, Paul Clements, David Garlan, Robert Nord, Bradley Schmerl, and Jaime R Silva. Documenting component and connector views with UML 2.0. Technical report, DTIC Document, 2004.

[1] http://jacop.osolpro.com/

[13] Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, April 2002.

[14] Jung Soo Kim and David Garlan. Analyzing architectural styles. *Journal of Systems and Software*, 83(7):1216–1235, 2010.

[15] Christian Koehler, Alexander Lazovik, and Farhad Arbab. Connector rewriting with high-level replacement systems. *Electronic Notes in Theoretical Computer Science*, 194(4):77–92, 2008.

[16] Jeff Kramer. Configuration programming — A framework for the development of distributable systems. In *CompEuro'90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, pages 374–384. IEEE, 1990.

[17] Christian Krause, Ziyan Maraikar, Alexander Lazovik, and Farhad Arbab. Modeling dynamic reconfigurations in Reo using high-level replacement systems. *Sci. of Comp. Prog.*, 76(1):23–36, 2011.

[18] Daniel Le Métayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, July 1998.

[19] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Anthony Tang. What industry needs from architectural languages: A survey. *Software Engineering, IEEE Transactions on*, 39(6):869–891, 2013.

[20] Anastasia Mavridou, Eduard Baranov, Simon Bliudze, and Joseph Sifakis. Configuration logics: Modelling architecture styles. In *Formal Aspects of Component Software — 12th International Conference, FACS 2015, Niterói, Brazil*, pages 256–274, 2015.

[21] Nenad Medvidovic and Richard N Taylor. A classification and comparison framework for software architecture description languages. *Software Engineering, IEEE Transactions on*, 26(1):70–93, 2000.

[22] OMG Unified Modeling Language (OMG UML) specification, Version 2.5. `http://www.omg.org/spec/UML/2.5/`. (Accessed on 17/03/2016).

[23] Mert Ozkaya and Christos Kloukinas. Are we there yet? analyzing architecture description languages for formal analysis, usability, and realizability. In *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, pages 177–184. IEEE, 2013.

[24] George A. Papadopoulos and Farhad Arbab. Coordination models and languages. *Advances in Computers*, 46:329–400, 1998.

[25] Dewayne E Perry and Alexander L Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[26] Grzegorz Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation*. World Scientific, 1997.

[27] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*, volume 1. Prentice Hall Englewood Cliffs, 1996.

[28] Michel Wermelinger, Antónia Lopes, and José Luiz Fiadeiro. A graph based architectural (re)configuration language. *ACM SIGSOFT Software Engineering Notes*, 26(5):21–32, 2001.

[29] Eoin Woods and Rich Hilliard. Architecture description languages in practice session report. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, pages 243–246. IEEE Computer Society, 2005.