Architecture Diagrams: A Graphical Language for Architecture Style Specification

Anastasia Mavridou  Eduard Baranov  Simon Bliudze  Joseph Sifakis
École polytechnique fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

Architecture styles characterise families of architectures sharing common characteristics. We have recently proposed configuration logics for architecture style specification. In this paper, 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.

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, 10]), coordination languages (e.g. [24, 1]) and configuration languages (e.g. [28, 15]). 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 types of the involved components and the topology induced by their coordination structure. Simple examples of architecture styles are Pipeline, Ring, Master/Slave, Pipes and Filters. For instance, Master/Slave architectures integrate two types of components, masters and slaves, such that each slave can interact only with one master. Fig. 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

\[
p_{1q1} + p_{2q2} \quad p_{1q2} + p_{2q1} \quad p_{1q1} + p_{1q2} \quad p_{2q1} + p_{2q2}
\]

Figure 1: Master/Slave architectures.

Architecture styles characterise not a single architecture but a family of architectures sharing common characteristics, such as the types of the involved components and the topology induced by their coordination structure. Simple examples of architecture styles are Pipeline, Ring, Master/Slave, Pipes and Filters. For instance, Master/Slave architectures integrate two types of components, masters and slaves, such that each slave can interact only with one master. Fig. 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

\[
p_{1q1} + p_{2q2} \quad p_{1q2} + p_{2q1} \quad p_{1q1} + p_{1q2} \quad p_{2q1} + p_{2q2}
\]

Figure 1: Master/Slave architectures.

To appear in EPTCS.
Architecture Diagrams

A Master/Slave architecture for two masters and two slaves can be represented as one among the following configurations, i.e. sets of connectors: \( \{p_1 q_1, p_2 q_2\}, \{p_1 q_2, p_2 q_1\}, \{p_1 q_1, p_1 q_2\}, \{p_2 q_1, p_2 q_2\} \). A term \( p_i q_j \) represents a connector between ports \( p_i \) and \( q_j \). The four architectures are depicted in Fig. 1. The Master/Slave architecture style denotes all the Master/Slave architectures for arbitrary numbers of masters and slaves.

We have recently proposed configuration logics \([19]\) 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 \([2]\), 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. Fig. 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 each connector. Degree \( d \) specifies the number of connectors in which each port instance is involved. The architecture diagram of Fig. 2 has a single connector motif involving generic ports \( p, q \) and \( r \).

A connector motif defines a set of possible configurations, where a configuration is a set of connectors. The meaning of an architecture diagram is a set of architectures that contain the union of all sub-configurations corresponding to each connector motif of the diagram. Fig. 3 shows the unique architecture obtained from the diagram of Fig. 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 \( r \). For \( p \), we have three instances and as both the multiplicity and the degree are equal to 1, each instance \( p_i \) has a single connector lead. For \( q \), 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. For \( r \), we have a single instance \( r_1 \) that has three connector leads to satisfy the degree constraint.

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, connectors from the sets obtained from step one are fused one by one, so that the multiplicities and the degrees of the ports are preserved, to generate the configuration of the connector motif.

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 also be intervals. Interval diagrams are strictly more expressive than simple 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 configurations of a connector motif; 4) 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.

A complete presentation, with proofs and additional examples, of the results in this paper can be found in the technical report [20].

The paper is structured as follows. Sects. 2 and 3 present simple and interval architecture diagrams, respectively. Sect. 4 presents an algorithm for checking conformance of diagrams. Sect. 5 discusses related work. Sect. 6 summarises the results and discusses 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. As in the previous section, 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: 1) a set of component types \( \mathcal{T} = \{T_1, \ldots, T_k\} \); 2) an associated cardinality function \( n : \mathcal{T} \rightarrow \mathbb{N} \), where \( \mathbb{N} \) is the set of natural numbers (to simplify the notation, we will abbreviate \( n(T_i) \) to \( n_i \)); 3) 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 \( \emptyset \neq a \subseteq \bigcup_{i=1}^{k} T_i.P \) 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 \( p \in a \).

Fig. 4 shows the graphical representation of a simple architecture diagram with a connector motif.

An architecture is a pair \( \langle \mathcal{B}, \gamma \rangle \), where \( \mathcal{B} \) is a set of components and \( \gamma \) 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 1. An architecture \( (\mathcal{B}, \gamma) \) conforms to a diagram \( (\mathcal{I}, n, \mathcal{C}) \) if, for each \( i \in [1, k] \), the number of components of type \( T_i \) in \( \mathcal{B} \) is equal to \( n_i \) and \( \gamma \) can be partitioned into disjoint sets \( \gamma_1, \ldots, \gamma_k \), such that, for each connector motif \( \Gamma_j = (a, \{m_p : d_p \}_{p \in a}) \in \mathcal{C} \) and each \( p \in a, 1 \) there are exactly \( m_p \) instances of \( p \) in each connector in \( \gamma_j \) and \( 2) \) each instance of \( p \) is involved in exactly \( d_p \) connectors in \( \gamma_j \).

We assume that, for any two connector motifs \( \Gamma_i = (a, \{m^1_p : d^1_p \}_{p \in a}) \) for \( i = 1, 2 \) with the same set of generic ports \( a \), there exists \( p \in a \), such that \( m^1_p \neq m^2_p \). Without significant impact on the expressiveness of the formalism, this assumption simplifies semantics and analysis. Details are provided in [20].

Multiplicity constrains the number of instances of the generic port that must participate in a connector, whereas degree constrains the number of connectors attached to any instance of the generic port. Consider the two diagrams and their conforming architectures shown in Figs. 5 and 6. They have the same set of component types and cardinalities. Nevertheless, their multiplicities and degrees differ, resulting in different architectures.

In Fig. 5 the multiplicity of generic port \( p \) is 1 and the multiplicity of generic port \( q \) is 3, thus, any connector must involve one instance of \( p \) and all three instances of \( q \). The degree of both generic ports is 1, so each port instance is involved in exactly one connector. Thus, the diagram defines an architecture with one quaternary connector.

In Fig. 6 the multiplicities of both generic ports \( p \) and \( q \) are 1. Thus, all connectors are binary and involve one instance of \( p \) and one instance of \( q \). The degree of \( p \) is 3, thus three connectors are attached to each instance. Thus, the diagram defines an architecture with three binary connectors.

![Figure 5: 4-ary synchronisation.](image1)

![Figure 6: Binary synchronisation.](image2)

### 2.2 Consistency Conditions

Notice that there exist diagrams that do not define any architecture. Let us consider the diagram shown in Fig. 4 with \( n_1 = 3, m_p = 1, d_p = 1, n_2 = 2, m_q = 1 \) and \( d_q = 1 \). Since the multiplicity is 1 for both generic ports \( p \) and \( q \), a conforming architecture must include only binary connectors involving one instance of \( p \) and one instance of \( q \). 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 thus, no architecture can conform to this diagram.

Consider a connector motif \( \Gamma = (a, \{m_p : d_p \}_{p \in a}) \) in a diagram \( (\mathcal{I}, n, \mathcal{C}) \) and a generic port \( p \in a \), such that \( p \in T.P \), for some \( T \in \mathcal{I} \). 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 \( m_p \) instances of \( p \) and no other ports and 2) each of the \( n_p \) instances of 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 generic port \( p \). Considering the diagram in Fig. 2 and the architecture in Fig. 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 \( p_1 q_1 q_2 \) \((i = 1, 2, 3)\), resulting in a configuration with three distinct connectors.

**Lemma 2.1.** Each regular configuration of a port \( p \) has exactly \( s_p \) connectors.
Prop. 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 not exceed 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, since the number of distinct connectors of a connector motif is bounded and equal to \( \prod_{q \in a} (n_q) \), there must be enough connectors to build a configuration. 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 if, 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) \( \forall q \in a, s_p = s_q \in \mathbb{N} \) and 3) \( s_p \leq \prod_{q \in a} (n_q) \).

### 2.3 Synthesis of Configurations

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

#### 2.3.1 Regular Configurations of a Generic Port

We start with an example illustrating the first step of the synthesis procedure for a port \( p \).

**Example 1.** Consider a port \( p \) with \( n_p = 4 \) and \( m_p = 2 \). There are 6 connectors of multiplicity 2: \( p_1p_2, p_1p_3, p_1p_4, p_2p_3, p_2p_4, p_3p_4 \), which 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 are shown in Fig. 7.

![Figure 7: Regular configurations of \( p \) with \( n_p = 4 \), \( m_p = 2 \).](image)

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

Consider Ex. 1 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, 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 for \( d_p = 1 \).

Table 1: Vector representation of regular configurations.

<table>
<thead>
<tr>
<th>( d_p = 1 )</th>
<th>( d_p = 2 )</th>
<th>( d_p = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>([00001 ]</td>
<td>([10001 ]</td>
<td>([11001] |</td>
</tr>
<tr>
<td>([01001 ]</td>
<td>([10101 ]</td>
<td>([20020] |</td>
</tr>
<tr>
<td>([00110 ]</td>
<td>([11110] | ([02200] |</td>
<td></td>
</tr>
<tr>
<td>([11111 ]</td>
<td>([120021 | ([210012 |</td>
<td></td>
</tr>
<tr>
<td>([01221] | ([021120 | ([030030] |</td>
<td></td>
<td></td>
</tr>
<tr>
<td>([102201 | ([201102] | ([00330] |</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 11
For $p$, 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. We have $GX = D$, 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 Ex. [1] the equations are:

$$\begin{align*}
    x_1 + x_2 + x_3 &= d_p, \\
    x_1 + x_4 + x_5 &= d_p, \\
    x_2 + x_4 + x_6 &= d_p, \\
    x_3 + x_5 + x_6 &= d_p,
\end{align*}$$

which is equivalent to

$$\begin{align*}
    x_1 + x_2 + x_3 &= d_p, \\
    x_3 &= x_4, \\
    x_2 &= x_5, \\
    x_1 &= x_6.
\end{align*}$$

(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 $a = \{p^1, \ldots, p^r\}$ have the same integer matching factor $s$. For each $p^j \in a$, let $\gamma^j = \{a^j_{i}\}_{i \in [1,s]}$ be a regular configuration of $p^j$. For arbitrary permutations $\pi_j$ of $[1,s]$, a set $\{a^j_{i} \cup \bigcup_{j=2}^{m} a^j_{\pi_j(i)}\}_{i \in [1,s]}$ 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 $p^j$. We denote by $w^j$ the dimension of the vector $X^j$.

In order to characterise the configurations of connectors conforming to $\Gamma$, we consider, for each configuration, the $v$-dimensional matrix $E = [e_{i_1, \ldots, i_v}]_{w^1 \times \ldots \times w^v}$ of 0-1 variables, such that $e_{i_1, \ldots, i_v} = 1$ if the connector $a^1_{i_1} \cup \ldots \cup a^n_{i_v}$ 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:

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

(2)

For instance, for a fixed $i \in [1,w^1]$, $e_{i_2, \ldots, i_v}$ describe all connectors that contain $a^1_i$. The regular configuration $\gamma^j$ is characterised by $X^j$, enforcing that $a^j_i$ belongs to $x^j_i$ connectors. The set of linear equations (2), combined with the sets of linear equations $G^jX^j = D^j$, for $j \in [1,v]$, fully characterises the configurations of $\Gamma$ and can be used to synthesise architectures from architecture diagrams.

**Example 2.** Consider a diagram $\{(T_1, T_2, n, \{\Gamma\})\}$, 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

$$\begin{align*}
    x_1 + x_2 + x_3 &= d_p, \\
    x_3 &= x_4, \\
    x_2 &= x_5, \\
    x_1 &= x_6, \\
    3y_1 &= d_q, \\
    y_1 &= y_2 = y_3 = y_4.
\end{align*}$$

(3)

Together with the constraints $x_i = \Sigma_j e_{i,j}$ and $y_j = \Sigma_i e_{i,j}$, for $E = [e_{i,j}]_{6 \times 4}$, equations (3) completely characterise all the configurations conforming to $\Gamma$.

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.
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 Fig. 10 graphically describes this style.

Example 5. 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 Ex. 4 there are no other connectors. The diagram of Fig. 11 graphically describes this architecture style.
2) a cardinality
8
Architecture Diagrams
3.1 Syntax and Semantics
varies from 0 to 2 and cannot be represented by an integer. m
of the masters might be optional, i.e. it might not interact with any slaves. In the first two architectures
nalities, multiplicities and degrees can be intervals. With simple architecture diagrams we cannot express
To enhance the expressiveness of diagrams we introduce interval architecture diagrams where the cardi-
Example 6. The diagram in Fig. 12 defines the set of architectures shown in Fig. 1. Notice that the degree
assumption greatly simplifies semantics and analysis.
To simple architecture diagrams, without significant impact on the expressiveness of the formalism, this
of generic port
a
with the same set of generic ports
p
different port instances of
p
interval the number of instances of
γ
sets
i
∈
An architecture
Semantics 2. An architecture ⟨B, γ⟩ conforms to an interval architecture diagram ⟨T, n, C⟩ if, for each
i ∈ [1, k], the number of components of type T_i in B lies in [n'_i, n''_i] and γ can be partitioned into disjoint sets
γ_1, . . . , γ_k, such that for each connector motif Γ_j = (a, {ty[m'_p, m''_p] : ty[d'_p, d''_p]}_{p ∈ a}) ∈ C and each
p ∈ a: 1) there are m_p ∈ [m'_p, m''_p] instances of p in each connector in γ_j; in case of a single choice
interval the number of instances of p is equal in all connectors in γ_j; 2) each instance of p is involved in
d_p ∈ [d'_p, d''_p] connectors in γ_j; in case of a single choice interval, the number of connectors involving an
instance of p is the same for all instances of p.
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 Γ_i = (a, {ty[m'_p, m''_p] : ty[d'_p, d''_p]}_{p ∈ a}) for i ∈ {1, 2},
with the same set of generic ports a, there exists p ∈ a such that [m'_p, m''_p]_1 ∩ [m'_p, m''_p]_2 = ∅. Similarly
to simple architecture diagrams, without significant impact on the expressiveness of the formalism, this
assumption greatly simplifies semantics and analysis.
Example 6. The diagram in Fig. 12 defines the set of architectures shown in Fig. 1. Notice that the degree
of generic port p is the multiple choice interval [0, 2], since one master component may be connected to

Figure 11: Multi-star architecture style.
two slaves, while the other master may have no connections. For the sake of simplicity, we represent
intervals \([x, x], mc[x, x]\) and \(sc[x, x]\) as \(x\).

**Proposition 3.1.** Interval architecture diagrams are strictly more expressive than simple architecture

## 3.2 Consistency Conditions

Similarly to simple diagrams, there are interval diagrams that do not define any architectures. Prop. 3.2
provides the necessary and sufficient conditions for the consistency of interval 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 max-
imum number of different connectors between these ports. These conditions are a generalisation of
Prop. 2.2.

Similarly to simple diagrams, there are interval diagrams that do not define any architectures. Prop. 3.2
provides the necessary and sufficient conditions for the consistency of interval 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 max-
imum number of different connectors between these ports. These conditions are a generalisation of
Prop. 2.2.

To simplify the presentation 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:

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

**Proposition 3.2.** An interval architecture diagram \((\mathcal{T}, n, \mathcal{C})\) 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 \( g^{a}_{p}, g^{d}_{p} \) such that, for \( |m^l_p|, m^u_p| = g^{a}_{p}(M_p) \) and \( |d^l_p, d^u_p| = g^{d}_{p}(D_p) \) hold:

1. \( m^l_p \leq n_p \) for all \( p \in a \), (where \( n_p = n_i \) for \( p \in T_i.P) \),
2. \( U \cap \bigcap_{p \in a}s_p \neq \emptyset \), where \( U = [1, \prod_{p \in a}\sum_{m=m_p}^{m^u_p}(n_p^u - n_p^l)] \), and

\[
s_p = \begin{cases} \left[ \frac{n_p^l d^l_p}{m^l_p}, \frac{n_p^u d^u_p}{m^l_p} \right] \cap \mathbb{N}, & \text{if } m^l_p > 0, \\ \left[ \frac{n_p^l d^l_p}{m^l_p}, \infty \right] \cap \mathbb{N}, & \text{if } m^l_p = 0. \end{cases}
\]

## 3.3 Synthesis of Configurations

The equational characterisation in Sect. 2.3 can be generalised, using systems of inequalities with some
additional variables, to interval architecture diagrams. Below, we show how to characterise the configura-
tions induced by \( n \) instances of a generic port \( p \) with the associated degree interval \( ty[d^l_p, d^u_p] \).

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 \( w = \binom{m^u_p}{m^l_p} \)) 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.

![Figure 12: Architecture diagram for architectures in Fig. 1](image-url)
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:

$$
\begin{align*}
    d_1 &= \cdots = d_n = d \\
    d_p^l &\leq d \leq d_p^u, \quad \text{for } ty = sc, \\
    d_p^l &\leq d_1 \leq d_p^u, \ldots, d_p^l \leq d_n \leq d_p^u, \quad \text{for } ty = mc.
\end{align*}
$$

\textbf{Example 7.} As in Ex. 1, consider a generic port $p$ and $n_p = 4, m_p = 2$. 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 $1 \leq d_i \leq 3, \text{for } i \in [1,4], x_1 + x_2 + x_3 = d_1, x_1 + x_4 + x_5 = d_2, x_2 + x_4 + x_6 = d_3, x_3 + x_5 + x_6 = d_4$.

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 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

\textbf{Example 8.} The diagram of Fig. 13 describes a particular Master/Slave architecture style and a conforming architecture for $n_1 = 2$ and $n_2 = 5$.

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.

\textbf{Example 9.} The diagram in the left of Fig. 14 describes the Repository architecture style involving a single instance of a component of type A and an arbitrary number $n_2$ of data-accessor components of type A. We require that all connectors involve the R component. In the right of Fig. 14 we show conforming architectures for $n_2 = 3$.

\textbf{Example 10.} The Map-Reduce architecture style [6] allows processing large data-sets, such as those found in search engines and social networking sites. Fig. 15 graphically describes the Map-Reduce architecture style. A conforming architecture for $n_1 = 3$ and $n_2 = 2$ is shown in Fig. 16.
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 workers (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 the Master. To this end, the in port of each RW is connected to the Rcontrol and read ports of the Master and some LFS, respectively. Each RW combines the results and writes them back to the GFS through a binary connector between their out and write ports.

## 4 Checking Conformance

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 $\gamma$ into $\gamma_1, \ldots, \gamma_t$ such that each $\gamma_i$ corresponds to a different connector-motif $\Gamma_i \in \mathcal{G}$ of the diagram; 3) for each connector motif $\Gamma_i$ and its corresponding $\gamma_i$, the number of times each port instance participates in $\gamma$ satisfies the degree constraints. The three statements correspond to functions VerifyCardinality, VerifyMultiplicity, and VerifyDegree, respectively. If all statements are valid the algorithm returns true, i.e. the architecture conforms to the diagram.

Function VerifyCardinality takes as input the architecture diagram $\langle \mathcal{T}, n, \mathcal{G} \rangle$ and the set of components $\mathcal{B}$ of the architecture $\langle \mathcal{B}, \gamma \rangle$. It counts the number of components for each component type in $\mathcal{B}$ and it returns true if for each component type $T$ of the diagram its cardinality matches the corresponding number of components in $\mathcal{B}$. Otherwise it returns false and algorithm 1 terminates.

Function VerifyMultiplicity takes as input the configuration $\gamma$ of the architecture $\langle \mathcal{B}, \gamma \rangle$ and the set
of connector motifs $\mathcal{C}$ of the architecture diagram $(\mathcal{B}, \gamma)$. The function checks whether there exists a partition of $\gamma$ such that each sub-configuration $\gamma_i$ of $\gamma$ corresponds to a distinct connector motif $\mathcal{C}_i$ of $\mathcal{C}$, i.e. each connector $k$ in $\gamma_i$ conforms to the multiplicity constraints of $\mathcal{C}_i$. If such a partition exists the function returns it. Otherwise, it returns $\emptyset$ and algorithm 1 terminates.

Function $\text{VerifyDegree}$ takes a connector motif $\Gamma$ of $\mathcal{C}$ and its corresponding sub-configuration of $\gamma$ assigned by $\text{VerifyMultiplicity}$. For each port instance in the sub-configuration it checks whether the number of times the port participates in different connectors is equal to the corresponding degree constraint of the connector motif. If the check fails, algorithm 1 terminates.

Algorithm 1 uses a number of auxiliary functions. Function $\text{generic}(p)$ takes a port instance and returns the corresponding generic port. Function $\text{typeof}(B)$ returns the component type of component $B$. Operation $\text{map}[\text{key}]++$ increases the value associated with the key by one if the key is in the map, otherwise it adds a new key with value 1.

5 Related Work

A plethora of approaches exist for architecture specification. Patterns [5, 9] 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.

A number of Architecture Description Languages (ADLs) have been developed for architecture specification [21, 29, 23]. Nevertheless, according to [18], architectural languages used in practice mostly originate from industrial development instead of academic research. Practitioners insist on using UML even though it lacks formal semantics. ADLs with formal semantics require the use of formal languages which are considered as difficult for practitioners to master [18]. To address this issue, we propose 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 re-
quired 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.

Alloy [12] has been used for architecture style specification, in the ACME [13] and Darwin [7] ADLs. The connectivity primitives in [13, 7] 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 [11, 22]. One important difference between connector motifs 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 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 [8, 17], 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, 14, 16], graph grammars can be used to define architecture styles: a style admits all the configurations that can be derived by its defining grammar. The use of context-free grammars allows inductive definitions and reasoning about architectures. The downside is that such definitions require additional non-terminal symbols to represent variable size structures, e.g. list of all slaves in a Master/Slave architecture. We take a different approach, whereby all constraints appear directly in the architecture diagram for which we provide denotational semantics. The rationale is the following: we assume that the reasoning is

```
Function VerifyMultiplicity(γ, C)
  Data: Configuration γ, set of connector motifs C
  Result: Returns a partition P of γ into connectors that satisfy the multiplicity constraint. If no partition exists, it returns ∅.
  // Map with key: generic port, value: number of port instances in connector
  for k ∈ γ do
    for p ∈ k do
      portscount ← { } for p, ∈ k do
        portscount[generic(p)] ++;
      x ← false;
      for a ∈ keys(portscount) do
        if a = false then
          for p ∈ a do
            if portscount[p] ≠ m then
              y ← true;
            break;
          if y then
            partition[Γ] ← partition[Γ] ∪ k;
            break;
          if x = false then
            return false;
    return true;

Function VerifyDegree(γ, Γ)
  Data: Configuration γ, connector motif Γ
  Result: Returns true if the degree requirements are satisfied. Otherwise, it returns false.
  // Map with key: generic port, value: number of connectors
  degrees ← { }
  for k ∈ γ do
    for p ∈ k do
      degrees[p] ++;
    for p ∈ keys(degrees) do
      if degrees[p] ≠ dgeneric(p) then
        return false;
  return true;
```
carried out by an “expert”, who defines the architectural style, whereas the “user” only needs the minimal information in order to select and instantiate it. Thus, structural information, e.g. necessary information for an inductive proof that the style imposes a certain property, does not appear in the diagram, but only the entities that form the target system.

6 Conclusion and Future Work

We studied architecture diagrams, a graphical language rooted in well-defined semantics for the description of architecture styles. We studied two classes of diagrams. Simple architecture diagrams express uniform degree and multiplicity constraints. They are easy to interpret and use but have limited expressive power. Interval architecture diagrams allow heterogeneity of multiplicity and degree and thus, are strictly more expressive. 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 using architecture diagrams to describe 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 paper with the JaCoP\(^1\) constraint solver. In the future, we plan to extend connector motifs with data flow information and study the expressive power of architecture diagrams.

References


\(^1\)http://jacop.osolpro.com/


