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.

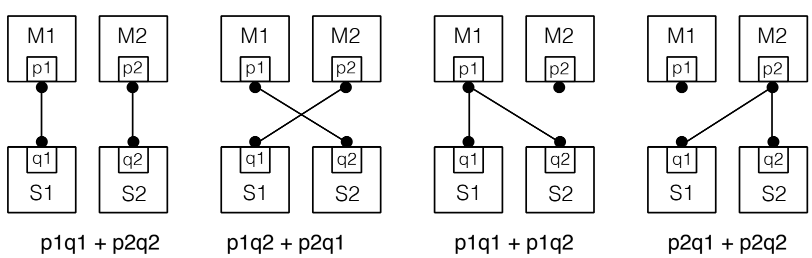

Figure 1: Master/Slave architectures.

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

To appear in EPTCS.

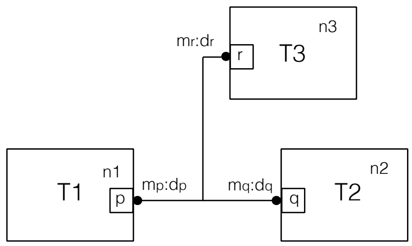

Figure 2: An architecture diagram.

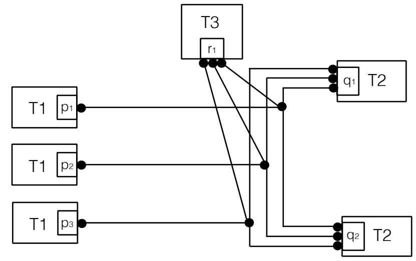

Figure 3: An architecture.

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 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

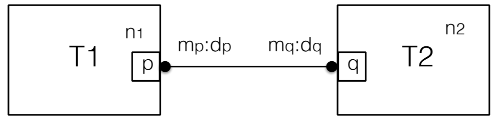

Figure 4: A simple architecture diagram.

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* $\langle \mathscr{T}, n, \mathscr{C} \rangle$ consists of: 1) a set of *component types* $\mathscr{T} = \{T_1, \dots, T_k\}$; 2) an associated *cardinality* function $n : \mathscr{T} \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers (to simplify the notation, we will abbreviate $n(T_i)$ to n_i); 3) a set of *connector motifs* $\mathscr{C} = \{\Gamma_1, \dots, \Gamma_l\}$ 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 $\langle \mathscr{B}, \gamma \rangle$, where $\mathscr{B}$ is a set of components and γ is a *configuration*, i.e. a set of connectors among the ports of components in $\mathscr{B}$. We define a connector as a set of ports that must interact. For a component $B \in \mathscr{B}$ 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 $B_1, \dots, B_n$ be all the components of type T in $\mathscr{B}$. For a generic port $p \in T.P$, we denote the corresponding port instances by $p_1, \dots, p_n$ and its associated cardinality by $n_p = n(T)$.

Semantics 1. An architecture $\langle \mathscr{B}, \gamma \rangle$ *conforms* to a diagram $\langle \mathscr{T}, n, \mathscr{C} \rangle$ if, for each $i \in [1,k]$, the number of components of type T_i in $\mathscr{B}$ is equal to n_i and γ can be partitioned into disjoint sets $\gamma_1, \dots, \gamma_l$, such that, for each connector motif $\Gamma_j = (a, \{m_p : d_p\}_{p \in a}) \in \mathscr{C}$ and each $p \in a$, 1) there are exactly m_p instances of p in each connector in γ_j and 2) each instance of p is involved in exactly d_p connectors in γ_j.

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 $m_p^1 \neq m_p^2$. 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.

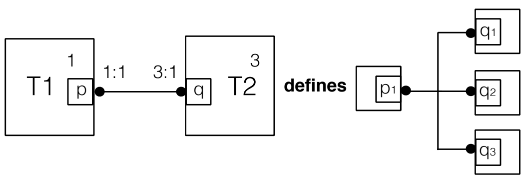

Figure 5: 4-ary synchronisation.

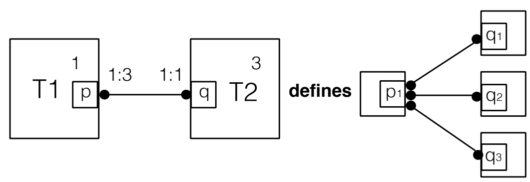

Figure 6: Binary synchronisation.

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 $\langle \mathscr{T}, n, \mathscr{C} \rangle$ and a generic port $p \in a$, such that $p \in T.P$, for some $T \in \mathscr{T}$. 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 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 $p_i q_1 q_2$ ($i = 1,2,3$), resulting in a configuration with three distinct connectors.

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

Table 1: Vector representation of regular configurations.

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

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 $\prod_{q\in a}\binom{n_q}{m_q}$, 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 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)* $\forall q \in a,\ s_p = s_q \in \mathbb{N}$ *and 3)* $s_p \leq \prod_{q\in a}\binom{n_q}{m_q}$.

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.

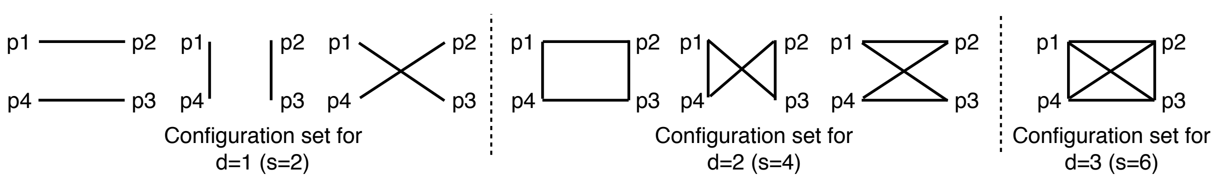

Figure 7: Regular configurations of p with $n_p = 4$, $m_p = 2$.

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 $d_p = 1$.

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 $D = [d_p, \dots, d_p]$ (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:

$$\begin{cases} x_1 + x_2 + x_3 = d_p\,, \\ x_1 + x_4 + x_5 = d_p\,, \\ x_2 + x_4 + x_6 = d_p\,, \\ x_3 + x_5 + x_6 = d_p\,, \end{cases} \quad \text{which is equivalent to} \quad \begin{cases} x_1 + x_2 + x_3 = d_p\,, \\ x_3 = x_4\,, \\ x_2 = x_5\,, \\ x_1 = x_6\,. \end{cases} \tag{1}$$

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 $a = \{p^1, \dots, p^v\}$ 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 $\{a_i^1 \cup \bigcup_{j=2}^{v} a_{\pi_j(i)}^j\}_{i \in [1,s]}$ 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 $G^j X^j = D^j$ 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 Γ, we consider, for each configuration, the v-dimensional matrix $E = [e_{i_1,\dots,i_v}]_{w^1 \times \dots \times w^v}$ of 0-1 variables, such that $e_{i_1,\dots,i_v} = 1$ if the connector $a_{i_1}^1 \cup \dots \cup a_{i_v}^v$ 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:

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

For instance, for a fixed $i \in [1,w^1]$, $e_{i,i_2,\dots,i_v}$ 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 set of linear equations (2), combined with the sets of linear equations $G^j X^j = D^j$, for $j \in [1,v]$, fully characterises the configurations of Γ and can be used to synthesise architectures from architecture diagrams.

Example 2. 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 $G_p X = D_p$, $G_q Y = D_q$ 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 $E = [e_{i,j}]_{6\times 4}$, 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

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

Figure 8: Matrix E with fixed values.

$$E \quad = \quad \begin{array}{c c} X \diagdown Y & \begin{array}{cccc} 1 & 1 & 1 & 1 \end{array} \\ \begin{array}{c} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{array} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 9: Valuation of matrix E.

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 3. Let us consider the diagram shown in Fig. 4 with $n_1 = 4$, $m_p = 2$, $d_p = 2$, $n_2 = 4$, $m_q = 3$ 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.

First, we compute the vectors X and Y that represent the regular configurations of generic ports p and q, respectively. Variables $x_1,\dots,x_6$ 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 $y_1,\dots,y_4$ 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.

Vector X can take one of the following values for $d_p = 2$: [110011], [101101], [011110], [200002], [020020] or [002200] (Ex. 1). Regular configurations of q are characterised by the equations $3y_1 = d$ and $y_1 = y_2 = y_3 = y_4$ (Ex. 2). 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 as shown in Fig. 8. Since, for all $i \in [1,6]$, we have $x_i = \Sigma_j e_{i,j}$ and, for all $j \in [1,4]$, we have $y_j = \Sigma_i e_{i,j}$, we can deduce that the only possible valuation for X, Y and E is the one shown in Fig. 9 corresponding to 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 4. 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.

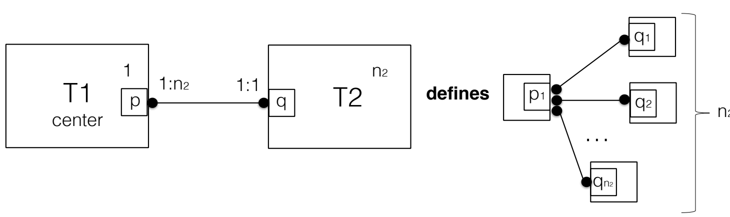

Figure 10: Star architecture style.

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.

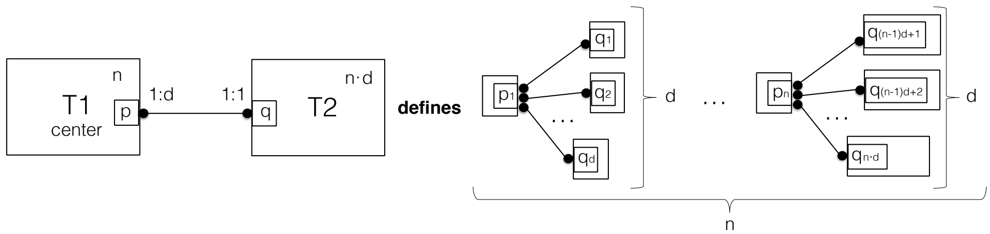

Figure 11: Multi-star architecture style.

3 Interval Architecture Diagrams

To enhance the expressiveness of diagrams we introduce interval architecture diagrams where the cardinalities, multiplicities and degrees can be intervals. With simple architecture diagrams we cannot express properties such as *"component instances of type T are optional"*. Let us consider the example of Fig. 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. In the first two architectures of Fig. 1 each master interacts with one slave, however, in the last two architectures one master interacts with both slaves while the other master does interacts with no slaves. In other words, the degree of m varies from 0 to 2 and cannot be represented by an integer.

3.1 Syntax and Semantics

An *interval architecture diagram* $\langle \mathscr{T}, n, \mathscr{C} \rangle$ consists of: 1) a set of *component types* $\mathscr{T} = \{T_1, \ldots, T_k\}$; 2) a *cardinality* function $n : \mathscr{T} \to \mathbb{N}^2$, associating, to each $T_i \in \mathscr{T}$, an interval $n(T_i) = [n_i^l, n_i^u] \subseteq \mathbb{N}$ (thus, $n_i^l \leq n_i^u$); 3) a set of *connector motifs* $\mathscr{C} = \{\Gamma_1, \ldots, \Gamma_l\}$ of the form $\Gamma = \Big(a, \{ty[m_p^l, m_p^u] : ty[d_p^l, d_p^u]\}_{p \in a}\Big)$, where $\emptyset \neq a \subseteq \bigcup_{i=1}^k T_i.P$ is a generic connector and $ty[m_p^l, m_p^u], ty[d_p^l, d_p^u]$, with $[m_p^l, m_p^u], [d_p^l, d_p^u] \subseteq \mathbb{N}$ 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 2. An architecture $\langle \mathscr{B}, \gamma \rangle$ *conforms* to an interval architecture diagram $\langle \mathscr{T}, n, \mathscr{C} \rangle$ if, for each $i \in [1, k]$, the number of components of type T_i in $\mathscr{B}$ 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 $\Gamma_j = \big(a, \{ty[m_p^l, m_p^u] : ty[d_p^l, d_p^u]\}_{p \in a}\big) \in \mathscr{C}$ 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_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 $d_p \in [d_p^l, d_p^u]$ 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 $\Gamma_i = \big(a, \{ty[m_p^l, m_p^u]_i : ty[d_p^l, d_p^u]_i\}_{p \in a}\big)$ for $i \in \{1, 2\}$, 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.

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

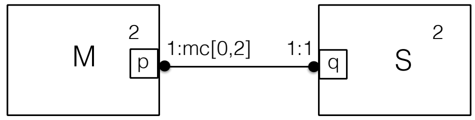

Figure 12: Architecture diagram for architectures in Fig. 1.

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 diagrams.*

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 maximum 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 $\mathscr{I}_T$ and $\mathscr{I}$ be the sets of, respectively, typed intervals and intervals, as in the definition of interval diagrams above. A function $g : \mathscr{I}_T \to \mathscr{I}$ 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 3.2. *An interval architecture diagram $\langle \mathscr{T}, n, \mathscr{C} \rangle$ is consistent iff, for each $T \in \mathscr{T}$, there exists a cardinality $n_i \in [n_i^l, n_i^u]$ and, for each connector motif $(a, \{M_p : D_p\}_{p \in a}) \in \mathscr{C}$ 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. $m_p^l \le n_p$*, 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* $U = \left[1, \prod_{p \in a} \sum_{m=m_p^l}^{m_p^u} \binom{n_p}{m}\right]$*, and*

$$s_p = \begin{cases} \left[\frac{n_p \cdot d_p^l}{m_p^u}, \frac{n_p \cdot d_p^u}{m_p^l}\right] \cap \mathbb{N}, & \textit{if } m_p^l > 0, \\ \left[\frac{n_p \cdot d_p^l}{m_p^u}, \infty\right) \cap \mathbb{N}, & \textit{if } m_p^l = 0. \end{cases}$$

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 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 $w = \binom{n_p}{m_p}$) 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.

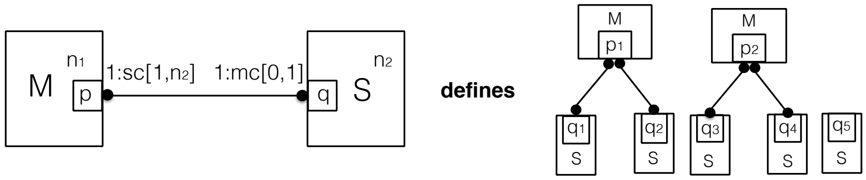

Figure 13: Master/Slave architecture style and conforming architecture.

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:

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

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 $1 \le d \le 3$, $x_1 + x_2 + x_3 = d$, $x_4 = x_3$, $x_5 = x_2$, $x_6 = x_1$. For the degree interval $mc[1,3]$ the corresponding constraints are $1 \le d_i \le 3$, for $i \in [1,4]$, $x_1 + x_2 + x_3 = d_1$, $x_1 + x_4 + x_5 = d_2$, $x_2 + x_4 + x_6 = d_3$, $x_3 + x_5 + x_6 = d_4$.

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 $n \in [n^l, n^u]$. 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 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.

Example 9. The diagram in the left of Fig. 14 describes the Repository architecture style involving a single instance of a component of type R 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 $n_2 = 3$.

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.

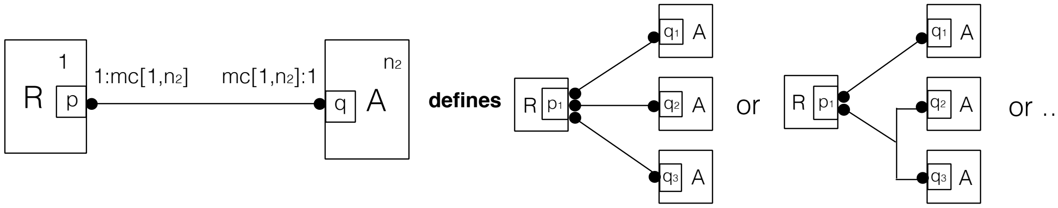

Figure 14: Repository architecture style and conforming architectures.

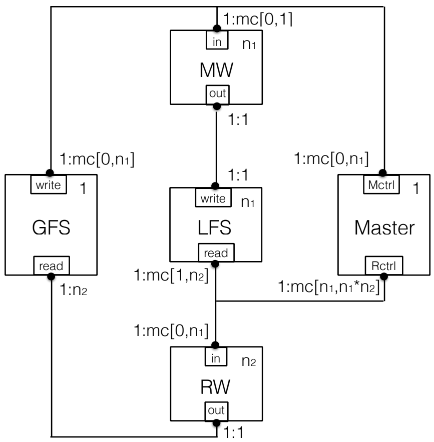

Figure 15: Map Reduce architecture style.

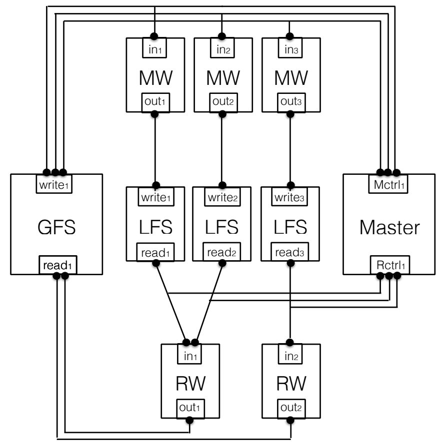

Figure 16: Map Reduce architecture.

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 with polynomial-time complexity checks whether an architecture $\langle \mathscr{B}, \gamma \rangle$ conforms to a simple diagram $\langle \mathscr{T}, n, \mathscr{C} \rangle$. It can be easily extended for interval diagrams as shown in [20].

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 $\Gamma_i \in \mathscr{C}$ of the diagram; 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 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.

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

Function VerifyMultiplicity takes as input the configuration γ of the architecture $\langle \mathscr{B}, \gamma \rangle$ and the set

Algorithm 1: VerifyArchitecture

Data: Architecture $\langle \mathscr{B}, \gamma \rangle$, diagram $\langle \mathscr{T}, n, \mathscr{C} \rangle$
Result: Returns *true* if the architecture satisfies the diagram $\langle \mathscr{T}, n, \mathscr{C} \rangle$. Otherwise returns $false$.

if not *VerifyCardinality*($\mathscr{B}$, $\langle \mathscr{T}, n, \mathscr{C} \rangle$) **then**
    return $false$;

$\mathscr{S}_\gamma \longleftarrow$ *VerifyMultiplicity*(γ, $\mathscr{C}$);
if $\mathscr{S}_\gamma = \emptyset$ **then**
    return $false$;

for $\Gamma \in \mathscr{C}$ **do**
    if *not VerifyDegree*($\mathscr{S}_\gamma[\Gamma], \Gamma$) **then**
        return $false$;

return *true*;

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

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

`/* Map with key: type, value: number of instances */`
countTypes $\longleftarrow \{\}$;
for $B_i \in \mathscr{B}$ **do**
    $countTypes[typeof(B_i)]++$;

for $T_i \in \mathscr{T}$ **do**
    if $countTypes[T] \neq n[T]$ **then**
        return $false$;

return *true*;

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

Function VerifyDegree takes a connector motif Γ of $\mathscr{C}$ and its corresponding sub-configuration of γ assigned by 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 `generic(p)` takes a port instance and returns the 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 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 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.

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

Data: Configuration γ, set of connector motifs $\mathscr{C}$
Result: Returns a partition $\mathscr{P}_\gamma$ of γ into connectors that satisfy the multiplicity constraint. If no partition exists, it returns $\emptyset$.

```
// Map with key: Γ, value: sub-configuration
partition ⟵ {}
for Γ ∈ C do
    partition[Γ] ⟵ ∅;

// Map with key: generic port, value: number
   of port instances in connector
for k ∈ γ do
    portscount ⟵ {} for p_i ∈ k do
        portscount[generic(p)] + +;
    x ⟵ false;
    for Γ = (a, {m_p : d_p}_{p∈a}) ∈ C do
        if a = keys(portscount) then
            y ⟵ true;
            for p ∈ a do
                if portscount[p] ≠ m_p then
                    y ⟵ false;
                    break;
            if y then
                partition[Γ] ⟵ partition[Γ] ∪ k;
                x ⟵ true;
                break;
    if x = false then
        return ∅;
return partition;
```

Function VerifyDegree(γ_i, Γ)

Data: Configuration γ_i, connector motif Γ
Result: Returns *true* if the degree requirements are satisfied. Otherwise, it returns false.

```
// Map with key: port, value: number of
   connectors
degrees ⟵ {}
for k ∈ γ_i do
    for p_i ∈ k do
        degrees[p_i] + +;
for p_i ∈ keys(degrees) do
    if degrees[p_i] ≠ d_{generic(p_i)} then
        return false;
return true;
```

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 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] Farhad Arbab (2004): *Reo: A channel-based coordination model for component composition*. Mathematical Structures in Computer Science 14(3), pp. 329–366, doi:10.1017/S0960129504004153.

[2] Paul Attie, Eduard Baranov, Simon Bliudze, Mohamad Jaber & Joseph Sifakis (2015): *A General Framework for Architecture Composability*. Formal Aspects of Computing, pp. 1–25, doi:10.1007/s00165-015-0349-8.

[3] Simon Bliudze & Joseph Sifakis (2008): *The Algebra of Connectors — Structuring Interaction in BIP*. IEEE Transactions on Computers 57(10), pp. 1315–1330.

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

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

[6] Jeffrey Dean & Sanjay Ghemawat (2008): *MapReduce: Simplified Data Processing on Large Clusters*. Commun. ACM 51(1), pp. 107–113, doi:10.1145/1327452.1327492.

[7] Ioannis Georgiadis, Jeff Magee & Jeff Kramer (2002): *Self-organising software architectures for distributed systems*. In: Proceedings of the first workshop on Self-healing systems, ACM, pp. 33–38.

[8] Dan Hirsch, Paola Inverardi & Ugo Montanari (1999): *Modeling software architectures and styles with graph grammars and constraint solving*. In Patrick Donohoe, editor: Software Architecture, IFIP 12, Springer, pp. 127–143.

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

[10] ISO/IEC/IEEE 42010 (2011): *Systems and software engineering — Architecture description*.

[11] James Ivers, Paul Clements, David Garlan, Robert Nord, Bradley Schmerl & Jaime R Silva (2004): *Documenting component and connector views with UML 2.0*. Technical Report, DTIC Document.

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

[12] Daniel Jackson (2002): *Alloy: A Lightweight Object Modelling Notation*. ACM Trans. Softw. Eng. Methodol. 11(2), pp. 256–290, doi:10.1145/505145.505149.

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

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

[15] Jeff Kramer (1990): *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, IEEE, pp. 374–384.

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

[17] Daniel Le Métayer (1998): *Describing software architecture styles using graph grammars*. IEEE Transactions on Software Engineering 24(7), pp. 521–533, doi:10.1109/32.708567.

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

[19] Anastasia Mavridou, Eduard Baranov, Simon Bliudze & Joseph Sifakis (2015): *Configuration Logics: Modelling Architecture Styles*. In: Formal Aspects of Component Software — 12th International Conference, FACS 2015, Niterói, Brazil, pp. 256–274.

[20] Anastasia Mavridou, Eduard Baranov, Simon Bliudze & Joseph Sifakis (2016): *Architecture Diagrams — A Graphical Language for Architecture Style Specification*. Technical Report 215210, EPFL. Available at: `http://infoscience.epfl.ch/record/215210`.

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

[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 & Christos Kloukinas (2013): *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, IEEE, pp. 177–184.

[24] George A. Papadopoulos & Farhad Arbab (1998): *Coordination Models and Languages*. Advances in Computers 46, doi:http://dx.doi.org/10.1016/S0065-2458(08)60208-9.

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

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

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

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

[29] Eoin Woods & Rich Hilliard (2005): *Architecture description languages in practice session report*. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA05), IEEE Computer Society, pp. 243–246.