Patterns in System Architecture Decisions

Abstract

This paper proposes a set of six canonical classes of architectural decisions derived from the tasks described in the system architecture body of knowledge and from real system architecture problems. These patterns can be useful in modeling architectural decisions in a wide range of complex engineering systems. They lead to intelligible problem formulations with simple constraint structures and facilitate the extraction of relevant architectural features for the application of data mining and knowledge discovery techniques. For each pattern, we provide a description, a few examples of its application, and briefly discuss quantitative and qualitative insights and heuristics. A few important theoretical properties of the corresponding set of patterns are discussed, such as completeness, degradedness, and computational complexity, as well as some practical guidance to be taken into account when applying them to real-life architecture problems. These patterns are intended to be a useful tool for researchers, practitioners and educators alike by: facilitating instruction and communication among system architects and with researchers from other fields such as combinatorics, computer science and operations research; and fostering reuse of domain-independent knowledge necessary to develop architecture decision support tools (and thus development time and cost reductions for such tools).

1. Introduction

1.1. Background and literature review

Crawley, Cameron and Selva [1, p. 110] provides the following definition for system architecture: "*The embodiment of concept, and the allocation of function to elements of form, and definition of relationships among the elements and with the surrounding context*". The role of the system architect is thus to make decisions about the main system concept, the main entities of function and form, and their relationships. We refer to these decisions as architectural decisions.

Since System Architecture is fundamentally a decision-making activity, much of the research in System Architecture has been around different methods of decision support. In particular, system architecture decision support can be broadly classified in three categories: a) *Descriptive* decision support, with architecture frameworks and applications of model-based systems engineering; b) *Prescriptive qualitative* decision support, primarily represented by architecture heuristics and patterns; c) *Prescriptive quantitative* decision support, such as in computation with Design Structure Matrices or tradespace exploration and optimization.

Descriptive Decision Support

Perhaps the majority of work in system architecture today is devoted to the development and application of system modeling methods and tools to represent, describe, document or communicate system architecture. The model-based paradigm is replacing document-centric practices to document system architectures; SysML [2], OPM [3], and FFBD and IDEF0 for functional architecture are among the most widely used tools [4].

A major development in the history of system architecture was the introduction of *architecture frameworks*. Essentially, a framework is a collection of views and viewpoints. A view is a model, diagram, table, picture or document that describes one or a few aspects of the system that are of relevance to one or more stakeholders, such as functional decomposition, concept of operations, or interfaces between the system components. A viewpoint is the template and set of guidelines to construct a view. While different frameworks have different goals, the purpose of frameworks is generally to improve the architecture process by providing a common language to facilitate communication across teams and codifying an institutionalizing best practices [5, p. 222].

While architecture frameworks are complex (e.g. DODAF 2.02 consists of 53 viewpoints) and systems modeling languages are expressive, they focus on describing a system with a predetermined design and architecture, and there is still little emphasis on the modeling of architectural and design decisions. The patterns presented in this paper and in particular their use in different graphical models can be seen as a formal way of defining architectural alternatives in model-based systems architecture that is more expressive than what exists today (e.g., SysML variants [2, pp. 128–130]).

Prescriptive Qualitative Decision Support

The foundational work that initiated the discipline of Systems Architecture is widely attributed to Rechtin [6] and Rechtin and Maier [5] later, who coined the term "systems architecting". This early work emphasized architecting as an *art*, rather than a science, and focused on the use of *heuristics*, i.e., rules of thumb containing simple principles or ideas that are meant to guide the architect in different phases of the architecture process (e.g., "*In partitioning, choose the elements so that they are as independent as possible, that is, elements with low external complexity and high internal cohesion.*"[5, p. 180]).

A related concept that is often used in the architecture of some specific types of systems is that of patterns. A *Pattern* in design can be defined as the description of a conflict or situation that appears often in the design of a specific kind of system, such as buildings or software systems, together with a prescriptive portion of how to solve that conflict. This is the definition that we adopt here. The value of patterns in design is two-fold: first, they facilitate communication between designers, since they can for example describe complex trade-offs succinctly by simply citing a known pattern; second, they foster reuse of knowledge and thus can reduce the time it takes to design a system. The first use of patterns in architecting is attributed to Christopher Alexander in the domain of civil architecture. Alexander's book "A Pattern Language: Towns, Buildings, Construction" contains a list of 253 patterns to design anything from houses and building to towns [7]. Each pattern contains a description of the problem, conflict or situation, as well as a proposed solution. For example, Pattern 159 "Light on each side of two rooms" states that "*When they have a choice, people will always gravitate to those rooms which have light on two sides, and leave the rooms which are only lit from one side unused and empty*". Hence, the Pattern suggests to "*Locate each room so that it has outdoor space outside it on at least two sides, and then place windows in these outdoor walls so that natural light falls into every room from more than one direction*". While the pattern-based approach to civil architecture was heavily criticized within the community, its impact has been undeniable in other domains, especially in software architecture [8].

The patterns presented in this paper intend to provide similar high-level qualitative, prescriptive, experience-based decision support to system architects, focusing on the process of modeling architectural decisions and formulating decision problems about the system architecture. Note that this does not consist in simply removing the need for a creative and expert system architect and reducing the architecture process to a mechanistic selection and application of patterns; instead, these patterns are meant to support the architect and help him or her make better decisions.

Prescriptive Quantitative Decision Support

Much of the prescriptive quantitative work in System Architecture has been around Design Structure Matrices (DSM), introduced by Steward in the 1960s [9] and widely adopted both in industry and academia due to their simplicity and modeling power [10]. DSMs are matrices that represent pairwise relationships between one or two sets of entities in a system, such as the structural relationships between the components of a system, or the mapping of system functions to components. While they can be used for purely descriptive purposes, DSMs are interesting because their symbolic nature yields itself to computation. For instance, clustering and sequencing algorithms can be applied to DSMs in order to help system architects find good system decompositions or good sequences of activities [11]. Of note, the use of binary matrices to represent and analyze the structure of systems was studied in depth by Warfield, Hill, Klir, Hall, Harary and Friedman among others during the efforts in the 1960's and 1970's to develop a general theory of

systems [12]–[15]. Their work represents the foundations of much of modern systems engineering theory, including precursors to DSMs, decision networks, and activity diagrams.

Another approach to quantitative decision support is based on the literature of decision analysis. Methods such as the Pugh matrix [16], the Analytic Hierarchy Process [17] and decision trees and networks [18] are applicable to architectural decision making, despite some well-known criticisms related to their rigor [19] or the obvious problem of scalability to large decision spaces. Optimization-based approaches are also common, in which the architecture problem is modeled through a set of variables (the architectural decisions) and a set of objectives (metrics that are proxies of stakeholder needs), and the goal is to find the combination of architectural variables that maximizes values to stakeholders (or at least an architecture that satisfices them [20]). Other essential tools used for conducting architectural trades, such as sensitivity analysis and design of experiments, were mostly adapted from the related fields of *Engineering Design* [21], [22] and *Multidisciplinary Design Optimization* (MDO) [23].

Relatively little work has been devoted to developing decision support tools that are applicable to any domain (e.g., automotive, aerospace), and especially tailored to the needs of system architects as opposed to lower-level design tasks. Dagli has a body of work on executable systems architecting using SysML and Petri nets [24]. Simmons developed the Architecture Decision Graph (ADG) in which the architectural decisions, alternatives, constraints and metrics are all represented in a single graph to show their logical relationships. Simmons and Koo developed a methodology to not only represent but also enumerate and evaluate architectures [25], using a modeling language based on OPM developed by Koo et al. [26].

Architecture space exploration and optimization tools are complex software packages and it takes substantial resources to develop and maintain them. The patterns presented in this paper are meant to improve system architecture by facilitating knowledge reuse, accelerating problem formulation, and reducing the costs of developing and maintaining architecture space exploration and optimization tools.

1.2 Preliminaries

The following basic terms from discrete mathematics are used in the remainder of the paper: Given two sets A and B, the Cartesian product of A and B is defined as $A \times B = \{(a, b)\ a \in A, b \in B\}$. A binary relation from A to B is any subset of the Cartesian product: $R \subseteq A \times B$. For example, if $A = [a_1, a_2]$ and $B = [b_1, b_2, b_3]$, then the following are valid binary relations: $R = \{(a_1, b_1), (a_1, b_2), (a_2, b_3)\}$, $R = \{(a_2, b_3)\}, R = \{\emptyset\}, R = A \times B$. N-ary relations are extensions of binary relations to $N > 2$, i.e. subsets of the Cartesian product $A_1 \times A_2 \times ... \times A_N$. A binary relation on a single set A is a binary relation from A to itself: $R \subseteq A \times A$. For example, the edges in a directed graph define a relation over the set of its vertices. A relation over a set A is said to be reflexive if every element of the set belongs to it, i.e. $(a, a) \in R\ \forall a \in A$. It is said to be symmetric if $(a, b) \in R \Leftrightarrow (b, a) \in R$ and antisymmetric if $(a, b) \in R, (b, a) \in R \Longrightarrow a = b$. It is said to be transitive if $(a, b) \in R, (b, c) \in R \Longrightarrow (a, c) \in R$. Two special kinds of relations on a set are of particular interest. A partial order relation is a relation that is reflexive, antisymmetric, and transitive. A set with a partial order relation is called a poset. For example, the real numbers and the operator "less than or equal to" define a poset. A total order relation is a partial order relation in which either $(a, b) \in R$ or $(b, a) \in R$ hold. Any permutation of a set of elements defines a total order relation. An equivalence relation is a relation that is reflexive, symmetric, and transitive. Informally, an equivalence relation describes elements that are similar (equivalent) according to some criterion. Elements that are equivalent are said to belong to the same equivalence class, and thus an equivalence relation partitions a set of elements into equivalence classes. For example, given the set of natural numbers up to 10, a possible equivalence relation may separate odd numbers from even numbers (i.e., in a sense, this relation states that all even numbers are equivalent and all odd numbers are equivalent, which can be seen as a description of modulo-

2 arithmetic.) Any partition of a set of elements (a grouping of its elements into mutually exclusive and collectively exhaustive subsets) defines an equivalence relation. See [13] for an excellent introduction to the theory of directed graphs to study systems structure.

1.3. Research gap analysis

Modeling of architectural decisions has so far embraced the traditional approach for formulating a problem used in decision analysis and design of experiments, in which an architecture a is represented as a set of decisions $X = \{x_1, x_2, \dots, x_n\}$, and each decision x_i has a discrete and usually small set of m_i alternatives O_i. The architecture space is thus simply defined by the Cartesian product of all the sets of alternatives.

$$\begin{gathered} a = [x_1, x_2, \dots, x_n]\ x_i \in O_i = \{o_{i1}, \dots, o_{im_i}\}\ \forall i = [1, n] \\ \mathcal{A} = O_1 \times O_2 \times \dots \times O_n, \text{ where } O_i \times O_j = \{(a, b) : a \in O_i, b \in O_j\} \end{gathered} \quad \textbf{(1)}$$

For example, consider the Apollo Project case study used by Simmons to illustrate his methodology. His formulation consists of $n = 9$ decisions, with a number of alternatives per decision ranging from 2 to 4, as shown in Table 1.

Table 1: Decisions and sets of alternatives for the Apollo program as formulated by Simmons [25, p. 101]

Decision	**Set of alternatives**
Earth Orbit Rendezvous (EOR)	{yes, no}
Earth Launch (EL)	{orbit, direct}
Lunar Orbit Rendezvous (LOR)	{yes, no}
Moon Arrival (MA)	{orbit, direct}
Moon Departure (MD)	{orbit, direct}
Crew Module Crew (CMC)	{2, 3}
Lunar Module Crew (LMC)	{0, 1, 2, 3}
Service Module Fuel (SMF)	{cryogenic, storable}
Lunar Module Fuel (LMF)	{cryogenic, storable, N/A}

One architecture in this case is obtained by choosing one alternative for each decision, and thus the full architecture space can be obtained by a full factorial enumeration algorithm, such as a set of nested *for* loops, one for each decision, or a mixed-radix generation algorithm [27]. However, some combinations of alternatives are invalid. Indeed, the first five decisions are coupled and concern what Simmons calls the mission mode of the architecture. For instance, one cannot launch direct to the Moon (EL="direct") and at the same time perform an Earth Orbit Rendezvous (EOR="yes"). This combination is logically impossible and thus must be eliminated from the architecture space by means of constraints. There are two other constraints of the same kind, which together eliminate 17 out of the 32 architecture fragments determined by the Cartesian product of the five sets of alternatives. Note that such constraints have nothing to do with the actual problem, they are just an artifact of the formulation (there are 15, and not 32 possible different mission modes). In fact, if we change the formulation to one that has a single mission mode decision with 15 alternatives, the need for these constraints disappears. Thus, how do we choose between the two formulations? On one hand, constraints generally add computational complexity to the optimization problem, since they can be seen as "dips" in the objective space that may undermine a local hill climbing process by the algorithm. Eliminating them will likely lead to better search performance. On the other hand, having a single decision for all the mission modes means that any information related to the sensitivity or coupling of the mission mode decisions is hidden. Thus, the architect may actually obtain more insight by using the formulation with more decisions.

While this example based on the mission-mode decision of the Apollo project illustrates a possible trade-off in the choice of formulation between computational efficiency and knowledge discovery, the decisions in the Apollo program can actually be easily modeled with a formulation based on Eq. (1). Indeed, this

approach works well for some types of architectural decisions, such as in specialization and characterization of the functions and components of the system (e.g. the type of propellant used for each stage of each vehicle in the system). However, it fails to adequately capture the structure of other important classes of architectural decisions, in particular decisions related to system decomposition and functional allocation– two tasks that are at the core of system architecture.

Consider for example the problem of architecting the US human space exploration program, tackled by Rudat et al. in [28]. A simplified functional analysis reveals that the main functions to be performed by the system are related to the transportation of humans and cargo (essentially, performing a series of propulsive maneuvers to travel between planetary bodies and conduct ascent and landing operations) and to habitation. One of the most important decisions to make in the system is the number of vehicles that it will have (system decomposition), and what functions each vehicle will do (mapping of function to form).

A formulation based on Eq. (1) could consist of a decision for the number of vehicles (e.g. 1, 2, 3 or 4 vehicles) plus one decision for each function that allocates it to one or more vehicles. This raises two difficulties. First, the set of alternatives for each function depends on the number of vehicles in the architecture. For example, if there are two vehicles, then each function (e.g. a core propulsive maneuver) has two or three alternatives: it can either be assigned to vehicle 1, to vehicle 2, or (perhaps) to both vehicles. It cannot be assigned to any subset of vehicles containing vehicles 3 and 4, which are absent from the architecture. Hence, a classical decision formulation would result in many dynamic constraints to eliminate the illogical cases – again, constraints that are an artifact of the problem formulation and could hinder both computational performance and knowledge discovery. Second, the set of alternatives for a function grows exponentially with the number of vehicles, which may also lead to computational and knowledge discovery difficulties.

Summarizing, current models used to formulate architectural decisions are based on explicit enumeration of the Cartesian product of the sets of alternatives for each decision. While this works well for some types of architectural decisions, it leads to limitations in both computational efficiency and knowledge discovery when applied to other important architectural decisions (e.g. system decomposition and functional allocation). Therefore, the following research question is raised:

Research Question: Is there a small set of mathematical models that combined can be used to formulate a wide range of system architecture problems in a way that is understandable, computationally efficient and leads to discovery of architectural insight?

2. Approach

We attempt to find one such set of mathematical models for the research question by adopting a two-fold approach: a) top-down, from the tasks of the system architect found in current systems architecture literature; b) bottom-up, based on generalization from a set of real-life specific system architecture problems.

Top-down approach from the tasks of the system architect

The definition of system architecture provided in Section 1 contains the essence of the main tasks of the system architect: defining the main elements of form and function and their relationships, with emphasis on the allocation of function to form. Many other definitions of system architecture exist: Ulrich and Eppinger define it as "*The arrangement of the functional elements into physical blocks*" [29, p. 165]; finally, the ISO/IEC 42010 standard describes it as "*The set of fundamental concepts or properties of the system and its environment, embodied in its elements, relationships, and the principles of its design and evolution*" [30]. While these definitions are clearly different, they all highlight a few tasks of system architecture, namely the arrangement of the system into subsystems and components, the mapping of functions to these components, and the definition of interfaces between the components.

A more detailed list of the tasks of the system architect can be found in all major textbooks in systems architecture (e.g., [1], [5], [6], [31], [32]) and includes: a) defining and prioritizing system goals; b) allocating goals to solution-neutral functions; c) specializing solution-neutral functions into solution-specific functions and decomposing functions into internal functions; d) defining functional flows and connectivity between functions; e) mapping or allocating function to form; f) aggregating entities of form (components) into subsystems; g) characterizing (i.e. defining the attributes of) entities of form; h) defining interfaces and connectivity between systems and subsystems; i) planning system deployment and operations.

We now focus on the aspects of these tasks that benefit from being formulated as decision making problems. Some of the previous tasks are mathematically very similar, such as allocation of goals to functions and function to form. If we combine mathematically akin tasks, we end up with 7 different tasks as showed in Table 2. Note that the numbers of these tasks do not necessarily imply a sequence.

Table 2: Tasks of the system architect that are amenable to formulation as decision making problems

Task	**Consists of**
1. **Decomposing**/Aggregating Form	Choosing a system decomposition, i.e. clustering elements of form.
2. **Mapping** Function to Form	Assigning elements of function to elements of form, or goals to functions
3. **Specializing** Form and Function	Choosing one among several alternatives for a certain element of Form or Function (e.g. going from solution-neutral to solution-specific)
4. **Characterizing** Form and Function	Choosing one among several alternatives for the attributes of an element of Form or Function
5. **Connecting** Form and Function	Defining system topology and interfaces
6. **Selecting** Goals and Functions	Defining scope by choosing among a set of candidate goals or functions
7. **Planning** system deployment and operations	Defining the sequence in which technologies will be infused into the project, system components will be deployed, or major operations will be conducted

The first task consists in the decomposition (or aggregation) of entities of form and function. Guidelines or heuristics to perform this task have been described in numerous works, such as [33], [34], [5, pp. 273–274], [32, pp. 218–220], [1, pp. 297–302]. Mathematically, this first task consists in finding an optimal partitioning of a set of entities.

The second task consists in mapping or allocating entities of one set (e.g. functions) to entities of another set (e.g. components). Note that the two tasks may not be independent, since function-to-form mapping may inform system decomposition. The analysis of function-to-form mapping from the point of view of functions and relations in set theory is the preferred approach in most of the literature [32, pp. 257–259], [35]. Mathematically, this task consists in defining an "optimal" binary relation between two sets of entities.

The third and fourth tasks concern the specialization and characterization of the main entities of function and form. Here, the terms specialization and characterization are used in the sense of OPM's structural relationships [3], also described in [1, p. 48]. Specialization refers to the concretization of a general entity into a more specific one, and typically, we will be concerned with the transition from solution-neutral to solution-specific entities of function and form. For example, one solution-neutral function of a weather satellite may be to take measurements of atmospheric humidity. A specialization of this function is to measure variations in the spectral radiance in specific portions of the infrared spectrum. Characterization refers to defining the value of an attribute of an entity of function or form. For example, two different characterizations of the humidity sounding function can be obtained by measuring in different parts of the

spectrum, such as infrared vs millimeter-wave. Mathematically, these two decisions are well suited for the formulation of Eq. (1).

The fifth task concerns the definition of interfaces between components, once these have been selected and defined. The connectivity task emphasizes the definition of topology in a system whose main decomposition of function and form has already been established, such as when deciding how to connect multiple fields with pipelines in a gas extraction system, or how to connect systems in a system-of-systems. This formulation appears in much of the early work in systems science and engineering [14], [36]–[38] and has been considered by some authors to be the primary formulation for system design problems [39]. Mathematically, the fifth task consists in finding the optimal subset of edges of a fully connected graph.

The sixth task concerns the selection of the system goals. Mathematically, the sixth task consists in finding the optimal subset among a set of entities.

Finally, the seventh task consists in planning system deployment and operations. Mathematically, this task may consist in finding the optimal ordering of a sequence of tasks.

In summary, in addition to the original formulation from Eq. (1), the following mathematical formulations emerge from the deductive analysis: choosing among all partitions of a set of elements, among all binary relations between two sets of entities, among all possible orderings of a set of entities, among all possible ways of connecting a set of entities, and among all subsets of a set of entities.

Bottom-up approach from examples of real-life architecture problems
In the bottom-up approach, we list several specific examples of real-life architectural decision problems and try to generalize the patterns that arise from them. We considered six different examples. These systems were chosen based on actual architecture studies conducted by the authors over the years. It takes substantial time and access to information and other resources to study the architecture of systems of this level of complexity. The description of these systems below is succinct for the sake of brevity, but interested readers can find more details about these systems and their decision formulations in our publications [40]–[44].

NEOSS: Architecting the Nation's Earth observing satellite systems: The major architectural decisions in Earth observing satellite systems are the selection of the instruments and the assignment of instruments into spacecraft and orbits [40]. For example, Envisat is a single-satellite mission in a sun-synchronous orbit at 800km carrying 10 instruments including a synthetic aperture radar, a scatterometer and multiple microwave and infrared imagers. Another important consideration is the sequence in which the different spacecraft are launched, since for example gaps in long data records must be avoided, and budget levels must be maintained.

NASA GN&C family of systems: An important consideration when designing a family of GN&C systems for NASA to be used in a wide range of missions from Class A man-rated missions to Class C robotic spacecraft is achieving the different required levels of reliability. Reliability can be traded against mass and cost by choosing the number and type of the sensors, computers and actuators, as well as the patterns of connectivity between them [42].

Communication systems: In the case of communication systems, some key architectural decisions concern the selection of protocols and network functionalities to implement, the allocation of such functions to system components, the number, type and location of assets (e.g. ground stations, satellites, balloons, aircraft), and the selection of frequency bands and technologies (e.g. radio-frequency vs optical communications) [43].

Energy systems: An important architectural decision of an energy system is the types of renewable and conventional energies that are used. Given certain projections of demand, and characteristics of different sources of energy, including efficiency, non-recurring and recurring cost, and reliability, choosing a

portfolio will drive key decisions such as whether to build new infrastructure requiring large capital expenditures. Other important decisions are related to the degree of intelligence of the power distribution networks [44].

Resource exploration systems: The architecture of a resource exploration system such as an oil platform can be modeled as a set of fields and reservoirs connected in an oil and gas network. The number, type and location of the facilities as well as their pattern of connectivity are the key decisions in this case [41].

When abstracted out of their domain-specific context, we observe that the decisions involved in these problems can be modeled using the formulations identified in the top-down approach: the original formulation in Eq. (1) is useful for example in the selection of frequency bands in communication systems or type of facility in resource exploration systems; the instrument packaging decision can be modeled as choosing among all partitions of a set of elements (instruments), or choosing among all binary relations between two sets of entities (instruments and orbits); the mission scheduling problem is essentially choosing among all possible orderings of a set of entities (missions); the decision of connecting fields and reservoirs in the oil and gas network is modeled as choosing among all possible ways of connecting a set of entities; finally, the instrument selection problem can be modeled choosing among all subsets of a set of entities.

One could argue that this small set of systems is not representative of any and every system out there. For example, all these systems can be described as either complex networked systems or systems of systems. In other words, these are systems for which major decisions are easy to interpret as graphs. For many other systems (e.g., cars or other vehicles), the use of graphs to describe decisions may be less natural or useful. However, in our experience, those are probably systems for which most of the architecture (system decomposition, mapping of function to form) is quite fixed (e.g. the internal combustion engine car), and most of the remaining decisions concern the characterization or specialization of function and form, which can be easily formulated using the simple combining pattern.

Review of classical combinatorial optimization problems
Appendix A contains a list of problems adapted from Gandibleux's classification of combinatorial optimization problems according to their combinatorial structure [45]. These problems were mostly used to check the completeness of our set of patterns and to make the connection between names and concepts in the patterns and the literature in combinatorial optimization.

3. Patterns in Architectural Decisions

The previous section has described the approach used to identify a set of mathematical models sought in the research question. This section proceeds to describe these formulations. A pattern-based approach is used: each model is introduced as a different pattern that appears when formulating architectural decision problems. The section starts by introducing and describing the six patterns: for each pattern, we provide a description and one or more examples. We then discuss some theoretical aspects of the set of patterns, particularly the completeness and degradedness of the mapping between real problems and patterns, and their computational complexity. Finally, some aspects related to using the patterns in practice are discussed, such as how to model the architecture of a complex system as a graph of interconnected decisions, each of which belongs to one of the patterns. A particular example concerning the architecture of the aforementioned NEOSS system is fleshed out in detail throughout the rest of the paper.

3.1. Description of the patterns

Table 3 introduces the six Patterns of architectural decisions. Each Pattern defines a different formulation of decisions that can be exploited to gain insight about the system architecture, and also to solve the problem more efficiently by using more appropriate tools. The term "architecture fragment" is used to refer to the outcome of making one or more decisions to define a part of the system architecture.

Table 3: The six Patterns of architectural decisions

Pattern	Description
Combining	There is a set of decisions where each decision has its own discrete set of options, and an architecture fragment is defined by choosing exactly one option from each decision.
Down-selecting	There is a set of candidate entities and an architecture fragment is defined by choosing a subset of it.
Assigning	There are two different sets of entities, and an architecture fragment is defined by assigning each entity from one set to a subset of entities from the other set.
Partitioning	There is a set of entities and an architecture fragment is defined by a partition of the set into subsets that are mutually exclusive and collectively exhaustive.
Permuting	There is a set of entities and an architecture fragment is defined by an ordering or permutation of the set.
Connecting	There is a set of entities that can be seen as nodes in a graph and an architecture fragment is defined by a set of edges in the graph.

The Combining Pattern

Description: Given a set of *n* decisions $X = \{x_1, x_2, \ldots, x_n\}$, where each decision x_i has its own discrete set of m_i options $x_i \in O_i = \{o_{i1}, \ldots, o_{im_i}\}$, an architecture or architecture fragment a in the Combining pattern is given by an n-tuple, i.e. the combination of exactly one option from each decision:

$$a = [x_1, x_2, \ldots, x_n]\ x_i \in O_i\ \forall i = [1, n] \quad \textbf{(2)}$$

Therefore, the Cartesian product of the sets O_i contains all the possible architectures in this pattern. This is the formulation used in design of experiments [46], decision trees, morphological analysis [47], and Simmons' Architecture Decision Graphs [25].

Example: The Apollo example presented in Section 1 is a good example of the Combining pattern, in which the architecture is represented by a set of 9 decisions, each with its own set options.

The Assigning Pattern

Description: The Assigning Pattern has to do with assignments or allocations between two pre-defined sets of entities, henceforth called for simplicity the *left set* and the *right set*. We say that the entities in the left set are assigned to the entities in the right set. In the most general formulation, each entity in the left set can be assigned to any subset of entities from the right set, including the empty set and the universal set. Given two sets of entities $L = \{L_1, \ldots, L_M\}$ and $R = \{R_1, \ldots, R_N\}$, an architecture or architecture fragment defined by an Assigning Pattern can be represented by an $M \times N$ binary matrix A where:

$$A_{ij} = \begin{cases} 1, & L_i \text{ connected to } R_j \\ 0, & \text{otherwise} \end{cases} \quad \textbf{(3)}$$

The pattern is pictorially illustrated using bipartite graphs and binary matrices in Figure 1. Note that this formulation includes any binary relation from L to R, i.e. any subset of $L \times R$. Variations of this formulation may include constraints on the minimum and maximum number of connections per entity of each set. Some cases are of particular interest. For example, *functions* are relations in which each element of L is assigned to exactly one element of R (i. e., not all relations are functions). In *one-to-one* or *injective* functions, each element of R is assigned to at most one element from L; in *onto* or *surjective* functions, each element of R is assigned to at least one element from L; finally, functions that are both injective and surjective are called *one-to-one correspondences* or *bijective* functions.

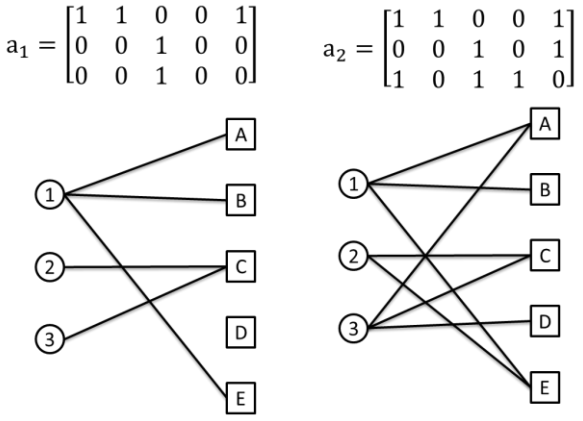

Figure 1: Two different architecture fragments in an Assigning Pattern with M=3, N=5

Example: Consider the architecture of the aforementioned Earth observing system in which 6 candidate instruments (sounder, radiometer, imager, radar, lidar and GPS receiver) and 3 orbits (geostationary, sun-synchronous and polar) have been selected. The mapping of instruments to orbits is an important architectural decision, since it defines the number and size of spacecraft, which drives most of the system's functionality, cost and performance [48]. In this particular case, there are no restrictions imposed on the relations between instruments and orbits, and thus the total number of architectures is simply given by $2^{18} = 262{,}144$. This includes architectures that fly multiple copies of some instruments, and zero copies of other instruments; In fact, it includes the architecture where no instruments are assigned to any orbit, i.e., the do-nothing alternative.

The Partitioning Pattern

Description: The Partitioning Pattern arises when there is a set of entities that need to be grouped into non-overlapping subsets. Given a set of entities $E = \{e_1, e_2, \dots, e_N\}$, an architecture fragment in the Partitioning Pattern is given by a partition *P* of the set U, i.e. any division of E into a number of non-overlapping subsets $\mathrm{P} = \{\mathrm{S}_1, \mathrm{S}_2, \dots, \mathrm{S}_\mathrm{m}\}$, where $S_i \subseteq \mathrm{U}, S_i \neq \{\emptyset\}\ \forall \mathrm{i}$, $1 \leq \mathrm{m} \leq \mathrm{N}$, and the subsets are mutually exclusive and exhaustive. In other words, P is a valid architecture if:

1. The union of all subsets in *P* is equal to *U*: $\bigcup_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{S_i} = \mathrm{E}$
2. The intersection of all elements in *P* is empty: $\bigcap_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{S_i} = \emptyset$

The Partitioning pattern is related to equivalence relations, since the blocks or subsets of a partition can be identified with equivalence classes. The pattern is illustrated in Figure 2. Architecture fragments in the Partitioning Pattern can be represented in multiple ways, but perhaps the most convenient one is a *restricted growth string*, i.e., an array of integers $[x_1, x_2, \dots, x_N]$ where $x_i = j$ means that element i is assigned to subset j and subset indices are assigned as needed. For example, in Figure 2, the first element is assigned to subset 1 and thus $a_1(1) = 1$. New subsets are created for the second and third elements as they are all in different subsets (labeled 2 and 3). The fourth element is assigned to subset 3. Elements 5, 6 and 7 are all assigned to a new subset labeled 4, and finally element 8 belongs to subset 1. The constraints $x_1 = 1, x_k \in \left[1, \max_{j<k} x_j + 1\right] \forall k$ are used to ensure that subsets are labeled uniquely, in ascending order as they are created by the assignment. Otherwise, as pointed out by [49], each real partition would have $N!$ possible encodings, which violates the principle of minimum redundancy of good representations [50].

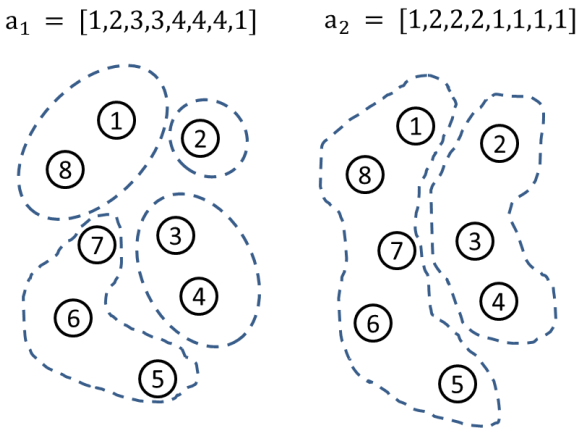

Figure 2: Two different architecture fragments in a Partitioning Pattern with 8 elements. In the array representation, $a(i) = j$ means that element i is assigned to subset j. Subset labels are assigned as needed.

A covering pattern can be defined as a derived case of the Partitioning Pattern in which the non-overlap constraint is relaxed. Thus, in the covering Pattern, an entity can be assigned to more than one subset. This formulation is frequently used in DSM Partitioning algorithms [51].

Example: The instrument-to-spacecraft assignment decision from the NEOSS example used to illustrate the Assigning Pattern can also be used as an example of the Partitioning pattern if a set of orbits is not available a priori. In this case, there is a set of 6 candidate instruments, and an architecture fragment is given by a partition of the 6 instruments into non-overlapping subsets. For instance, two alternative architecture fragments for this problem are {{lidar}, {radar, radiometer}, {imager, sounder, gps}} and {{lidar}, {radar}, {radiometer, gps}, {imager, sounder}}. Note that there is exactly one copy of each instrument in the architecture, and that no orbit is defined for each of the resulting spacecraft – these orbits would be chosen based on the characteristics of each payload.

The Down-selecting Pattern

Description: The Down-selecting pattern is motivated by the tasks of the system architect that require choosing a subset of among a set of candidate entities. Typically, this pattern arises in situations in which each entity adds some value and costs some resources and there are limited resources, so that choosing all entities is not possible. The Down-selecting pattern is similar to the well-known 0-1 knapsack problem in operations research [52]. Given a set of entities $E = \{E_1, \dots, E_N\}$, an architecture fragment in the Down-selecting pattern is given by an $N \times 1$ binary array A where:

$$A_i = \begin{cases} 1, & E_i \text{ selected} \\ 0, & \text{otherwise} \end{cases} \quad \textbf{(4)}$$

When the Down-selecting pattern arises in system architecture it is usually linked to defining the *scope* of the system, in the sense of a deliberate trade-off of capabilities of the system against cost/risk in the presence of non-linear mappings of capabilities to stakeholder utility and capabilities to cost. For example, a Down-selecting Pattern can be used to describe the problem of choosing among a set of conflicting system goals, or among a set of candidate assets with overlapping capabilities.

Example: Continuing with NEOSS example, choosing among a set of candidate instruments, without regards for the spacecraft or orbits in which they will be allocated, is an instance of the Down-selecting pattern. For example, if we choose from a set of 8 candidate instruments {radiometer, altimeter, imager, sounder, lidar, GPS receiver, synthetic aperture radar (SAR), spectrometer}, two different architecture fragments from a Down-selecting Pattern would be $A_1 = \{\text{radiometer,altimeter,GPS}\} = [1,1,0,0,0,1,0,0]$, $A_2 = \{\text{imager, sounder, SAR, spectrometer}\} = [0,0,1,1,0,0,1,1]$.

The Connecting Pattern

Description: The Connecting Pattern emphasizes the connectivity between a predetermined set of entities. Given a set of entities, seen as the vertices of a graph, an architecture fragment in the Connecting Pattern is given by a set of edges connecting those vertices. Hence, the Pattern can be visualized as adding edges to an empty graph, or equivalently, choosing a subset of edges from a fully connected graph. More formally, given a set of entities $E = \{E_1, \dots, E_N\}$, an architecture or architecture fragment defined by a Connecting Pattern is given by an $N \times N$ binary square matrix A where:

$$A_{ij} = \begin{cases} 1, & E_i \text{ connected to } E_j \\ 0, & \text{otherwise} \end{cases} \quad \textbf{(5)}$$

Therefore, the architecture fragment is represented by a binary square matrix, namely the adjacency matrix of the graph. Note that the matrix is only symmetric if the graph is directed. Figure 3 shows two different architecture fragments in a generic instance of the Connecting Pattern with six entities.

$$a_1 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad a_2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

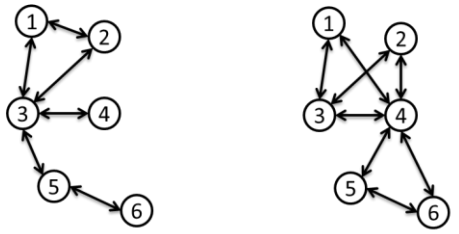

Figure 3: Illustration of the Connecting Pattern

The Connecting Pattern is closely related to the definition of system interfaces using DSM [10], and to the definition of system connectivity in Wymore's classical textbook [53, p. 111]. It is also close to the formulation of general design problems by Chapman et al. [39]. Furthermore, the Connecting pattern is related to a number of well-known problems in graph theory, such as edge cover or max flow problems.

Example: The Connecting Pattern appears naturally in networked systems where resources such as data, water, oil, or energy are transported throughout the network. In these cases, the Connecting Pattern has to do with the topology of the network, which is an important architectural decision in networks (e.g., star vs meshed networks). For example, consider the problem of architecting the water distribution network of a small city. The city is divided in several areas of different characteristics and needs (e.g., population, presence of natural resources, water needs). Assuming that a number of water generation or treatment plants are already present in different points in the city, the decision of how to connect those plants is an instance of the Connecting Pattern.

The Permuting Pattern

Description: The Permuting Pattern appears when the architectural decision consists in defining a one-to-one mapping of a set of entities to a set of positions (of equal size). More formally, given a set of entities $E = \{E_1, \dots, E_N\}$, an architecture fragment in the Permuting Pattern is given:

$$O = \{x_1, x_2, \dots, x_N\} : x_i \in [1; N] \forall i, x_i \neq x_j \forall i, j \in [1; N] : i \neq j \quad \textbf{(6)}$$

The Permuting Pattern can thus be seen as a bijection of E onto itself. An architecture fragment in the

Permuting Pattern can be represented by an array of integers, with two possible interpretations: element-based, or position-based. In the element-based representation, the value of entry i indicates the index of the element in in the ith position, whereas in the position-based representation, the value of entry i indicates the position of the ith element. For example, the sequence {element 2, element 4, element 1, element 3} can be represented by the array $O = [2,4,1,3]$ (element-based representation), or by the array $O = [3,1,4,2]$ (position-based representation).

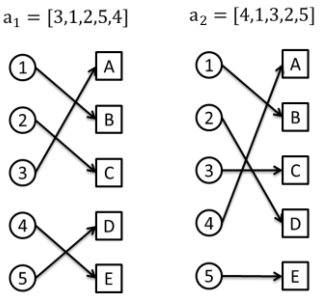

Figure 4: Illustration of the Permuting Pattern (element-based representation is used in the arrays)

The Permuting Pattern is related to the concept of DSM sequencing in functional or process architecture [54], in which the entities to be sorted are functions or operations in a process. It is also related to two classical combinatorial optimization problems: the Traveling Salesman Problem [55] and the Job Scheduling Problem [56].

In addition to its important role in functional architecture, the Permuting Pattern can also support the Physical Architecture process (i.e. the process of describing the hierarchy and connectivity of physical components in the system), for instance when dealing with the high-level geometrical positioning or arrangement of a set of components. Such considerations may in some cases be an important driver of cost and performance, and therefore worthy of considering early, at the architecture level as opposed to the detailed design level.

Example: In 2009, NASA cancelled the "Constellation" space exploration program due primarily to budgetary restrictions. To replace Constellation, a committee of experts led by Norm Augustine proposed several alternative strategies, including one known as the Flexible Path. In the Flexible Path architecture, an important decision is the sequence of destinations (e.g., the Moon surface, an asteroid, the Lagrange points) on the way to Mars. In the NEOSS example, choosing the launch sequence for the missions is an instance of the Permuting pattern.

3.2. Theoretical aspects of the patterns

Completeness

As in any other classification scheme, the question of completeness arises: can any system architecture problem be expressed using the six patterns provided? If this question is approached literally from the perspective of pure theoretical possibility, then the answer is yes, since for example any architectural decision problem can be formulated as an integer decision problem using the Combining pattern. Perhaps it is less intuitive to see that any decision problem can also be seen as an instance of a Connecting Pattern in which there is a bipartite graph with two types of nodes (decisions and options) and a subset of the possible edges must be selected. In fact, we prove in the next section that one can reduce problem instances from one pattern to any of the other patterns, which implies that any of the six Patterns would satisfy the completeness property.

However, a less theoretical but arguably more useful completeness question would be whether there exist problems for which none of the patterns is appropriate. This must take into account attributes such as how

natural or how *difficult* it is to formulate the problem using this set of patterns. Part of these attributes may be expressed in objective terms. For example, the number of domain-independent constraints that need to be added to a pattern in order to capture the mathematical structure of the problem can be used to measure how natural the formulation is. A scientific approach to answering this question would require conducting experiments involving real system architects formulating real architecting problems, with and without the patterns. This is expanded upon in the future work section.

Degradedness
In addition to completeness, a reasonable question to ask is that of degradedness, i.e., whether or not the mapping between real problems and patterns is one-to-one. As we have already mentioned, this is not the case, since a given problem can often be expressed using several patterns. In fact, we proceed to prove that a problem instance of any pattern can be reduced into an instance of any other pattern.

Some of these relationships are intuitive. For example, a Down-selecting problem can be seen as a Combining problem where all decisions are binary. An Assigning problem can be seen as a set of Down-selecting problems, one for each element of the left set. The same is true for a Connecting problem, which can be seen as a Down-selecting problem, with one binary decision per edge. The Permuting and Partitioning Patterns can both be seen as special cases of the Assigning Pattern in which additional constraints are added. In fact, it is relatively easy to see how all patterns can be reduced into instances of the Combining problem and the Down-selecting problem.

Other relationships are less intuitive. For example, it is hard to see a priori how a Partitioning problem can be reduced to a Permuting problem. One possible formal proof consists in proving that all patterns can be reduced into a Down-selecting problem, which is simple, and then proving that the Down-selecting problem can be reduced to all patterns, which is slightly harder. To see the latter, consider the following: We have already seen that a Down-selecting problem is an instance of a Combining problem with all binary variables. Similarly, a Down-selecting problem is an instance of a Connecting problem with one edge per element, and an instance of an Assigning problem with two elements in the right set. The two hard cases are Partitioning and Permuting. We use results from combinatorics to achieve this result. A bi-objective (e.g., cost-performance) down-selecting problem can be reduced to a number of single-objective Down-selecting problems where the goal is to maximize performance at a given cost. Such problem can be reduced to the problem of deciding whether given a set of weights, some subset of them adds up to a certain integer number C exactly. A basic result from combinatorics is that this can be reduced to a set partitioning decision problem, in which one must decide whether a set of weights can be partitioned into two sets of equal sum [57]. Finally, the set partition decision problem can be readily reduced to a Partitioning problem (q.e.d.). One can show that a Down-selecting problem can be reduced into a Permuting problem by a similar reduction path through the knapsack problem and the traveling salesman problem.

Computational Complexity
In the analysis of the computational complexity of problems, problems are classified according to the time and/or space that it would take the best algorithm to find a solution assuming a certain model of computation (e.g., Deterministic Turing Machine, Non-Deterministic Turing Machine). It turns out that most classical combinatorial optimization problems are in NP, which means that a solution can be found in polynomial time in a hypothetical non-deterministic Turing machine – in other words, no polynomial time algorithm is known on a real machine. If a problem H is such that all problems in NP can be reduced to H in polynomial time, then H is said to be NP-hard. If a problem is both in NP and NP-hard, then it is said to be NP-complete. Most of the combinatorial optimization problems mentioned in this paper are actually NP-complete. This was proved by Karp using a hierarchy of reductions between the different problems down to Satisfiability [57], [58].

Since these classical problems can be reduced to instances of the patterns, it follows that by reduction the patterns are also NP-complete. More precisely: the set partitioning problem can be reduced to a problem of

the Partitioning pattern where there are is a single metric with no interactions between subsets; the traveling salesman problem can be reduced to a Permuting problem with a single metric without interactions; the generalized assignment problem can be reduced to an Assigning problem; the 0/1 knapsack problem can be reduced to a Down-selecting problem with a single metric and no interactions; the min edge cover problem can be reduced to a Connecting problem with a single element; the integer programming problem can be reduced to a Combining problem with linear metrics and constraints. Since all these classical problems are known to be NP-complete, it follows that all patterns are NP-complete.

3.3. Using the patterns in practice

Mapping of patterns to the tasks of the system architect
Some of the relationships between the six Patterns and the tasks of the systems architect have already been highlighted. For example, it is very apparent that Task 1 Decomposing Form and Function is most closely related to the Partitioning Pattern. Table 4 is a DMM providing a more exhaustive account of this mapping between system architecture tasks and Patterns: '1' indicates the primary assignment, and '2' indicates a secondary assignment. Table 4 shows how the patterns can be used by practitioners and students as a means to support their efforts to create mathematical formulations of real life architecture problems.

Table 4: DMM illustrating the mapping between architecture tasks and patterns.

	Combining	Down-selecting	Assigning	Partitioning	Permuting	Connecting
1. Decomposing Form and Function				1		2
2. Mapping Function to Form			1	2		
3. Specializing Form and Function	1	2				
4. Characterizing Form and Function	1	2				
5. Connecting Form and Function					2	1
6. Selecting Goals		1	2			
7. Planning deployment and operations				2	1	

As stated, the system decomposition task is mostly related to the Partitioning pattern, since decomposition/aggregation can be seen as grouping entities into an undetermined number of non-overlapping subsets. The use of the Connecting Pattern for studying decomposition may also be appropriate when using DSM decomposition techniques, but it is less natural. Task 2, mapping of function to form, usually implies that a set of functions is available. A set of entities of form may or may not be initially available. When it is available, or at least the number of entities of form has been decided, then the task is most naturally related to the Assigning Pattern, where the left set contains the functions and the right set contains the entities of form. When a predetermined set of entities of form is not available, then a formulation based on a Partitioning pattern is more appropriate, where functions are simply grouped into any number of subsets that will later become the entities of form. In Tasks 3 and 4, if a discrete and small set of options is available for each entity of function or form to be specialized or characterized, then the most closely related pattern is the Combining Pattern. Both tasks are also assigned to the Down-selecting pattern, to recognize that both the specialization and characterization processes may include choosing a subset among a set of candidate functions or attributes (e.g., the frequencies at which a measurement is

taken). Task 5, connecting form and function, is primarily linked to the Connecting Pattern when it has to do with defining the connectivity of a set of entities, which can be modeled as the edges of a graph. The Permuting pattern is selected as a secondary option, since the connecting task may also involve deciding the arrangement of components, i.e. an assignment of components to positions that is well modeled by the Permuting Pattern. Task 6, selecting goals, is assigned to the Down-selecting Pattern, since it is assumed that goals are chosen from a set of initial possibly conflicting goals. The Assigning Pattern is selected as a secondary choice to cover the case where goals are prioritized and classified into 3 or more classes as in Kano analysis [59] (e.g., must have, should have, could have). Finally, Task 7, planning of system deployment and operations, is most naturally formulated as an instance of the Permuting Pattern, and the Partitioning Pattern is chosen to acknowledge that some operations should be performed in parallel, i.e. clustered.

Reading Table 4 by columns instead of by rows suggests that, while the Combining Pattern is the one used by most decision support frameworks in systems architecture, it may not be the most natural formulation for most key architecting tasks, such as mapping function to form, finding a system decomposition or defining interfaces between components.

Mapping of patterns to combinatorial optimization problems
Table 5 provides a mapping between the Patterns presented in this paper and classical optimization problems.

Table 5: DMM mapping classical combinatorial optimization problems to the most similar architecture tasks

	Combining	Down-selecting	Assigning	Partitioning	Permuting	Connecting
Integer programming	X					
0/1 Knapsack		X				
Generalized assignment			X			
Weapon-target assignment			X			
Set partitioning				X		
Integer partitioning				X		
Job scheduling					X	
Traveling Salesman					X	
Min Edge cover						X
Spanning tree						X

Given Table 5, one could be tempted to conclude that the formulation of these classical optimization problems is identical to that of the corresponding Patterns. However, when the complete formulation is considered, including a value function and constraints if appropriate, important differences appear that need to be resolved before mixed-integer programming techniques can be used. These differences concern especially two aspects: the number of metrics (single vs multi-objective) and the additivity (sum or multiplicative) assumptions in the objective functions. Indeed, most formulations of classical combinatorial optimization problems are single-objective, whereas most realistic system architecture problems have two or more metrics. For example, in the classical set partitioning problem, the goal is to minimize the cost of a partition, given by the sum of the costs of its subsets. In real-life architecture problems, having at least two metrics, such as performance and cost, may be very important to understand the trade-offs behind the decision. One may partially alleviate this problem by combining the metrics using for example a weighted

average criterion or a lexicographic approach, but this solution relies on strong a priori assumptions about the preferences of the decision makers [45].

Furthermore, traditional formulations of combinatorial optimization problems of interest to the operations research community have strong assumptions of additivity of the value function to make them tractable. For example, the classical 0/1 knapsack problem assumes that each item has two known values, a benefit and a cost, and that the value and cost of a subset of items is the sum of the values and costs of the items. This additivity is exploited as the Bellman condition and an efficient dynamic programming formulation is used to solve them. The value functions and constraints that are typically used in system architecture do not satisfy this additivity property, due for example to the presence of redundancies in capabilities across assets or diminishing marginal utility curves, and performance synergies and interferences between components. Going back to the Earth observing example, if we already have an instrument providing measurements of atmospheric temperature, the utility of adding another instrument that also provides temperature measurements is lower due to this redundancy.

It is important to note that many tricks exist in the operations research community to accommodate for certain non-linearities and constraints. However, their application usually requires an expert in operations research, which is not always available in system architecture teams. Furthermore, these tricks don't cover the general case of a blackbox value function.

Problem Decomposition
Real-life architecture problems will often need to be decomposed into a hierarchy of decisions before any of the patterns can be used. This is apparent from the fact that the system architecture process consists of several tasks, which require more than one pattern to be formulated. Furthermore, there might be dependencies between the decisions at different levels, so that the set of alternatives available for a decision at a certain level depends on the alternative chosen for a parent decision.

In the NEOSS example, the architecture is modeled by 3 sets of decisions: 1) instrument selection (an instance of the Down-selecting pattern), 2) instrument to spacecraft assignment (an instance of the Partitioning pattern), and 3) mission scheduling (an instance of the Permuting pattern). In this case, the instrument to spacecraft assignment decision must logically occur after the instrument selection decision, because we need to know how many instruments there are to allocate before we can actually encode the instrument to spacecraft assignment decision as a partitioning decision. Similarly, the mission scheduling decision can only be made after the instruments have been assigned to spacecraft to create missions.

Another example concerns a family of Guidance, Navigation and Control (GN&C) systems for the NASA Constellation Program. In this case, the architecture is modeled by a number of sensors, computers and actuators chosen from a catalog of components and connected in any possible topology that respects the overall flow of data from sensors to computers to actuators. This can be modeled as a number of Combining decisions and Assigning decisions. First, the number of components (number of sensors, computer and actuators) are chosen (Combining). Once the number of components has been chosen, the Combining decisions concerning the types of components can be made. Finally, the two Assignment decisions modeling the connections between sensors and computers, and computers and actuators can be made.

4. Knowledge reuse for fast problem formulation

The Patterns presented in this paper are valuable to practitioners because they facilitate quantitative decision support in several ways. First, they help system architects find adequate formulations for their architecture problems by reducing the complexity and ambiguity of the formulation problem into the much better defined problem of defining hierarchy of decisions and choosing the pattern for each decision. Second, it facilitates communication between system architects and other stakeholders by establishing a common vocabulary that is accessible to all of them. Third, and perhaps most importantly, it fosters the reuse of domain-independent, pattern-specific knowledge from project to project. For instance, code implementing

different evolutionary operators or local search techniques in partitioning or assigning spaces, or knowledge concerning distance functions that work better in each pattern are examples of knowledge that is domain-independent to a certain extent and specific to a pattern, and thus can be reused from project to project.

Table 6 identifies a few categories of knowledge that can be reused and some specific examples for all patterns. The categories from Table 6 are the following:

- *Counting all alternatives*: All patterns have known analytical formulas to count all the alternatives represented by the pattern. For example, a partitioning decision with N elements defines $Bell(N)$ alternatives, where $Bell()$ indicates Bell numbers. This knowledge is important because counting the size of an architecture space allows the system architecture analyst to estimate the total time that it would take to evaluate the entire architecture space and thus decide if full factorial enumeration is possible or if he/she must resort to partial exploration of the space.
- *Generating all alternatives*: Many combinatorial algorithms exist to generate all alternatives represented by a decision. For example, the 2^{MN} alternatives defined by an assigning decision of sizes M, N can be enumerated by simply counting in binary from 0 to $2^{MN} - 1$ keeping a constant length of MN bits, and reshaping the resulting arrays into $M \times N$ matrices.
- *Generating random alternatives*: Generating all alternatives is sometimes not possible if the space is too large. In this and other cases, generating a random sample of alternatives is useful. Many algorithms exist for enumerating random n-tuples, partitions, and permutations. Many rely on pseudo-random number generation techniques such as the linear congruential method. For example, a random sample of partitioning architectures can be generated by sampling the number of subsets in the partitions from an exponential distribution and then using the linear congruential method to generate random sequences of integers assigning the elements to the bins.
- *Common architectural features*: We define architectural features as an extension of the idea of architectural variables that includes any combination of variables using logical or arithmetic operators. For example, in the partitioning pattern, whether or not two particular elements i and j are in the same subset (i.e. $x_i == x_j$) is an architectural feature. Such architectural features can be used to define constraints or as feature set to train a data mining algorithm.
- *Distance functions*: Distance functions are used by many statistical algorithms that need to assess the similarity between two architectures. The Manhattan norm is usually an appropriate distance for the patterns based on binary decisions, but other specialized distances are more appropriate for partitioning and permuting decisions.
- *Local search and evolutionary operators*: Finally, it is possible to reuse pattern-specific knowledge related to the operators used in local search and evolutionary optimization frameworks. For example, while the traditional single-point crossover operator might be a perfectly adequate choice for many binary decisions, it is a poor choice in most partitioning and permuting problems.

Note that this paper is not meant to be a library of such pattern-specific knowledge, and therefore Table 6 is absolutely not complete. Its purpose is rather to show the amount of knowledge that such library would help architects reuse from problem to problem.

Finally, Table 6 provides solutions that can be applied to individual canonical decisions. As stated earlier, the application of this knowledge to a real problem given by independent decisions is straightforward, but as the dependencies becomes more complex, this knowledge recombination can become challenging. For instance, counting how many architectures there are in the NEOSS example requires carefully taking into account the dependencies between decisions: for 6 instruments, and using a Partitioning pattern without predefined orbits, there are $\sum_{n=0}^{6}\binom{6}{n}\sum_{k=0}^{n} S(n,k)k! = 9{,}366$ unique architectures, which is much smaller number than $2^6 \cdot Bell(6) \cdot 6!$ which would result in the case of independent decisions.

Table 6: Examples of pattern-specific knowledge that can be reused

Pattern-specific task	**Combining**	**Down-Selecting**	**Assigning**	**Partitioning**	**Permuting**	**Connecting**
Counting all alternatives	$\prod_{i=1}^{n} m_i$	2^N	2^{MN}	$B_n = \sum_{k=0}^{n} \left\{ {n \atop k} \right\}$ $\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$	$N!$	$2^{\frac{N(N-1)}{k} + mN}$ k=1 if directed k=2 if undirected m=1 if self-connections allowed m=0 otherwise
Generating all alternatives	Mixed-radix generation [27, p. 282]	Binary-radix generation	Binary-radix generation	Restricted growth strings [27, pp. 415–440]	Lexicographic tree expansion [27, p. 319]	Binary-radix generation
Generating random alternatives	Linear congruential method [60, p. 10]	Linear congruential method [60, p. 10]	Linear congruential method [60, p. 10]	Urn model [61]	Random walk from initial permutation	Linear congruential method [60, p. 10]
Architectural features	Propositional logic	Binary schemata	Specific element-bin matches, specific empty bins, max #elems per bin	Specific elements are together, separate, alone #bins	o-schemata [62] specific elements by the beginning/end, or before, after or between other elements	Binary schemata Presence of hubs Trees
Distance function	Manhattan norm Lexicographic distance	Manhattan norm	Manhattan norm	Transfer distance [63]	Kendall-tau distance	Manhattan norm
Local Search and evolutionary operators	Single-point crossover	Single-point crossover	Single-point crossover	2-exchange, Cycle exchange Cycle crossover	Swap, Interchange, Reverse Partially matched crossover	Single-point crossover

5. Qualitative decision support: Architectural Insight through Pattern-specific heuristics

Alexander's concept of patterns intended to provide qualitative decision support to practitioners confronting recurring problems in civil architecture. This is done in a context-specific environment such as designing buildings or software. In system architecture, many domain-independent heuristics and tactics have been proposed related to the tasks of the architect mentioned in Section 2 (e.g. [5]). Can the same thing be done for the patterns presented in this paper? For instance, given an architect confronted with an assigning decision, is there a set of heuristics, tactics, principles or guidelines that the architect can use to make the decision independently of whether the task is related to assigning instruments to orbits, or functions to components? We argue that in all but the simplest pattern (Combining), this is possible, and furthermore it leads naturally to the discussion of architecture styles. Some of this insight is discussed below.

Down-selecting Pattern

The Down-selecting pattern is about finding subsets of elements that "work well together" compared to others, as determined by the net effect of the interactions between the elements in the subset. Due to these interactions, among other things, the value of a subset of elements is not equal to the sum of the values of the elements. We designate these interactions as *synergies* when: a) new emerging capabilities arise from the interaction between the elements in the subset and none of the original elements had those capabilities; or b) no new capabilities emerge, but the performance with which the subset can perform a certain capability improves by virtue of the interaction. We designate these interactions as *interferences* when the performance of some of the capabilities of the elements is degraded by the presence of the other elements in the subset. Finally, there is sometimes *redundancy* in the capabilities of elements in a subset. Consider the aforementioned NEOSS decision of selecting from a set of remote sensing instruments. The radar altimeter and microwave radiometer instruments are highly synergistic elements, because the accuracy of the altimetry measurement obtained from the radar improves in the presence of the radiometer thanks to a lower error due to the atmosphere. Conversely, the synthetic aperture radar and the lidar have negative interactions, because they both require large amounts of power and have very different orbit requirements. Furthermore, there might be some redundancy between the radar altimeter and the lidar, since both can be used to do topographic measurements.

The piece of preescriptive advice is thus that when confronted with a problem matching the Down-selecting Pattern, the architect must consider the net effect of the interactions (synergies, interferences) between the candidate elements as well as the redundancy in their capabilities. Generally speaking, good architectures in the Down-selecting Pattern are those that lead to high synergy, low interference, and low redundancy.

Assigning Pattern

Given two sets of entities, the Assigning pattern is concerned with mapping elements from one set to elements of the other set, and it is fundamentally about deciding how connected or sparse we want this mapping to be. In the NEOSS example, the connections are instrument-orbit assignments. orbits. Assigning an instrument to an orbit means adding a "connection" to the architecture. In this example, connections (i.e. instruments in orbit) can thus be costly. On the other hand, in the GN&C example, connections between sensors and computers (and computers to actuators) are cables (and the corresponding software and interfaces) which can be much cheaper. Adding connections may improve system properties such as data throughput or reliability, but it usually requires an investment of complexity and cost.

While mapping instruments to orbit and connecting sensors to computers are fundamentally different problems, we see that they are decisions with similar features. In both cases, if we enforce that each element from the left set must be assigned to at least one element of the right set (an additional constraint that is not present in the most general formulation of the pattern), we can define two "extreme" architectures at the boundaries of the architectural tradespace: 1) a "channelized architecture" where each "left" element is matched to exactly one "right" element; 2) a "fully cross-strapped" architecture where every "left" element

is connected to every "right" elements [42]. We will refer to these extremes as architecture styles, the channelized *style* vs the fully cross-strapped *style* (see Figure 5).

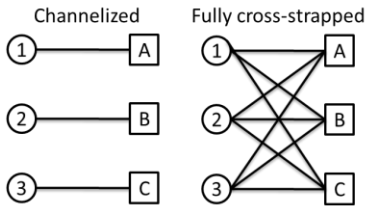

Figure 5: Channelized vs fully cross-strapped architecture styles in the Assigning pattern

We argue that the channelized vs cross-strapped trade-off applies to both function to form mapping, and connectivity of form and function. In the case of mapping function to form, the channelized architecture corresponds to a perfect satisfaction of Suh's independence axiom [34], since each function is performed by one element of form, and each element of form performs only one function. In the case of connecting components, the trade-off is basically one between throughput and reliability versus cost. The "fully cross-strapped" style of architectures may lead to more complex and thus most costly architectures than the channelized style due to a higher number of connections, but at the same time it may lead to more reliable architectures thanks to increased redundancy. Note however that this increase in redundancy may be diminished by the presence of common-cause failures, since strong coupling between entities may allow for failures to propagate through the system.

Therefore, the prescriptive advice for the architect is to study the relative cost of adding connections between elements compared to the cost of the elements themselves, and weigh that against the potential increases in reliability, which should consider common-cause failures and the cost/reliability utility functions of the decision maker.

Partitioning Pattern

The Partitioning Pattern is concerned with grouping a set of entities into non-overlapping and collectively exhaustive subsets, and it is fundamentally about the degree of centralization we want in our architecture. Hence, we define two new architecture styles: centralized (or monolithic) and decentralized (or distributed) architectures. In monolithic architectures, all elements are grouped in a central "large" bin, whereas in fully distributed architectures, all elements are assigned to "smaller" dedicated bins (see

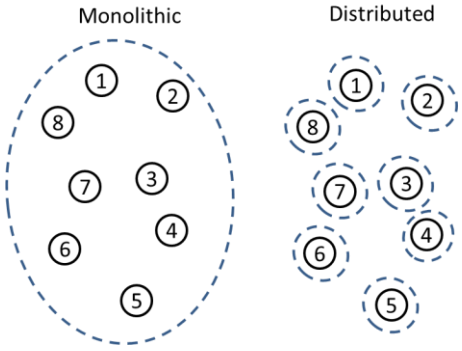

Figure 6).

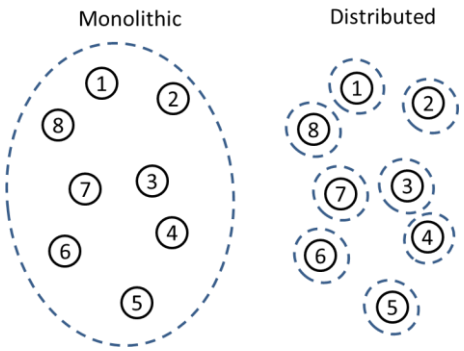

Figure 6: Monolithic vs distributed architecture styles in the Partitioning pattern

Interactions between elements, namely the synergies and interferences introduced in the context of the Down-Selecting Pattern, play a key role in the Partitioning Pattern. In the NEOSS example, if the radar altimeter and the radiometer are put on different spacecraft and on different orbits in the Partitioning problem, it will very hard to obtain the benefits of their synergistic interaction. Indeed, the instruments need to be either on the same spacecraft, or close enough to allow for cross-registration. Similarly, electromagnetic interference will only occur if two instruments are on the same spacecraft or close enough.

Thus, partitioning decisions will usually have an impact on cost and performance. More generally, this trade-off is also related to system properties such as evolvability, robustness, and flexibility. The monolithic or centralized style leads to architectures with very few (but complex) components, which may take longer to develop and may be costlier. It minimizes redundancy if there is common functionality needed by all elements in the bin, thereby decreasing total cost, but for the same reason it may decrease reliability and robustness due to presence of single points of failure, and may lower performance of individual elements due to competition for resources (e.g., power) in the bins. It captures the positive interactions between components, which can increase performance, but also the negative interactions, which may actually increase cost. On the other hand, a distributed approach leads to more elements and thus lower cost per element, but potentially higher total cost. It may lead to increased reliability and robustness, and increased performance due to tailored design of each bin to its single element. Synergies may be lost which can decrease performance, but interferences are also avoided, which may actually reduce cost.

Therefore, the prescriptive piece of advice for architects facing partitioning decisions is to look at: a) the relative effects of synergies and interferences between elements on system performance and cost; b) the amount of functionality that needs to be replicated on every bin of the partition, and the relative cost of this common functionality compared to the cost penalty due to interferences; c) the willingness to pay of the decision maker to improve programmatic aspects such as reducing development risk, shortening development time, maintaining a flat expense profile over time, robustness to component failures and ease to introduce evolution.

Connecting Pattern

The Connecting pattern is related to defining the connectivity between the nodes of a network. Therefore, we borrow the classical network architectures as architecture styles for the Connecting pattern, namely: bus/star architecture, ring architecture, mesh architecture, and tree architecture. These styles are illustrated in Figure 7.

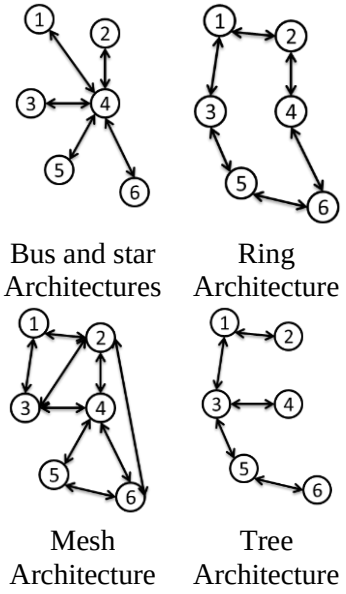

Figure 7: Set of 6 elements connected according to different architecture styles in the Connecting pattern

In both bus and star architectures, also known as Hub-and-Spoke architectures, there is a central element (the hub or the bus) to which all elements in the network are connected. In the case of bus architectures, the bus is not a real node in the sense that it doesn't perform any functions other than relaying data, whereas the hub in star architectures performs all node functions such as generating or absorbing traffic flows in the network. In ring architectures, every node is connected to exactly two nodes in such a way that the traffic in the network flows in a single closed circuit – the ring. In mesh architectures, all nodes perform relaying functions and nodes are connected in arbitrary complex patterns. Finally, in tree architectures, there is a hierarchy of nodes such that each node is only connected to nodes in levels immediately above or below its own level in the hierarchy.

The choice between these styles depends strongly on the cost of connections and is mostly related to trade-offs between cost, latency, reliability, robustness, and scalability. For example, bus/star architectures are usually affordable and scalable but they have a single point of failure in the bus/hub. Ring architectures have good latency, but are unreliable and not very scalable. Meshed architectures are good for latency, reliability and robustness but are more costly. Finally, tree architectures are very scalable and maintainable, but they are poor in terms of robustness.

Permuting Pattern

The Permuting pattern is about defining an ordering of a set of entities, i.e. a one-to-one mapping of a set of N entities onto a set of integer numbers from 1 to N. These numbers can usually be interpreted as lying on a continuous axis such as time, space or others, so they often have absolute meanings in addition to relative meanings. Moreover, these entities have some cost and benefit attached to them that depend on their absolute and relative positions in the sequence. In particular, it is often the case that cost/benefit decrease/increase monotonically (e.g. through a discount rate) with the relative or absolute position in the sequence. Furthermore, the cost and benefit of some entities may depend on whether or not they are before/after or between other entities (interactions). Finally, a budget is sometimes defined in the form of a constraint dictating the maximum cost that can be spent per unit of the continuous axis (time or space). The permuting pattern is thus fundamentally about maximizing the total discounted benefit and minimizing total discounted cost, while balancing the budget. Given this view of the pattern, we can define three extreme strategies as styles for the permuting pattern: in the *greedy* style, high benefit/high cost items are all put at the beginning of the sequence, whereas in the *incremental* style, high benefit/high cost items are all put at the end of the sequence. These styles are illustrated in Figure 8.

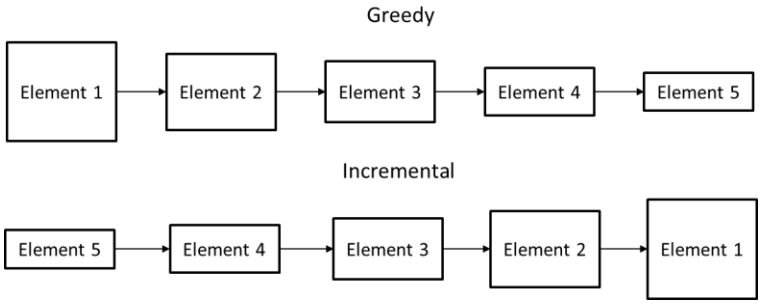

Figure 8: Greedy vs incremental architecture styles in the permuting pattern

Generally speaking, the greedy style aims to obtain undiscounted benefits at the price of paying undiscounted costs, whereas the thrifty style aims at delaying/discounting costs at the price of discounting benefits. The incremental approach is more desirable from the perspective of "time to value" since, in the greedy approach, no value is delivered until all costs of the largest element have been incurred at the speed given by the budget. Moreover, in the presence of uncertainty, and if information is gained along the time axis, the incremental approach is also preferable from the perspective of flexibility or evolvability, since delaying the deployment of "large system elements" gives the architects the option to change the system architecture to adapt to changes in the system or the environment. In summary, the choice between greedy and incremental solutions in the permuting pattern has to do with the relative discount rates of cost and benefit, the decision makers' utility functions for cost and benefit as well as their willingness to pay for lifecycle properties such as flexibility or evolvability.

6. Discussion and Conclusion

Intellectual Contributions

This paper has introduced six patterns that appear in the structure of architectural decisions: Combining, down-selecting, assigning, partitioning, permuting, and connecting. Each pattern consists of a descriptive and a prescriptive part. The descriptive provides a model and formulation of the situation they represent, plus one or more examples. The prescriptive part is two-fold: it provides quantitative decision support in the form of knowledge (algorithms) that can be used to develop architecture tradespace exploration tools, and qualitative decision support in the form of heuristics and rules of thumb that the system architect must consider when facing a decision of a given pattern.

In addition to that, some theoretical and practical aspects of the patterns are discussed. From the theoretical standpoint, it has been shown that all patterns lead to NP complete problems, and that one can transform a problem formulated in a given pattern to any other pattern. From the practical standpoint, the mapping between the patterns and the tasks commonly considered to be the responsibility of the system architect is discussed.

Impact

We argue that the introduction of these patterns has strong impacts in the research, practice and education of system architecture. In particular, this work:

- May help **researchers** by bridging the gap between mostly disconnected bodies of research in system architecture (descriptive, prescriptive and qualitative, prescriptive and quantitative).
- May help **practitioners** by providing a methodology that simplifies the formulation process from an open-ended creative process to a well-defined configuration problem, in which there a finite number of types of decisions that can be connected in a graph. Furthermore, it fosters the reuse of pattern-specific, domain-independent knowledge which may lead to saving time and resources in the formulation of system architecture problems and development of the corresponding tools.

- May help **students** by connecting quantitative aspects with qualitative aspects thus expanding their zone of proximal development [64]. Indeed, one of the challenges in the education of systems architecture is that the body of knowledge in the field contains both high-level qualitative concepts that are hard to absorb by students with little professional experience (e.g. the principles and heuristics), and lower-lever quantitative concepts that are hard to absorb by more experienced professionals that haven't been in contact with mathematical tools for years (e.g. combinatorics, optimization, statistics). These patterns can be an anchor for both types of students to better grasp the concepts that are harder for them to learn.

Limitations and Future work

Some of the limitations of this work are related to the challenge of defining a set of patterns as the canonical patterns in architectural decisions. These limitations have already been discussed in the body of the paper, such as the non-uniqueness of the set of patterns, or the non-unicity of the mapping of real-life architectural decision to patterns.

In the application of the patterns as descriptive decision support, one of the major current limitations lies on the ability to model architectural decisions in model-based environments such as SysML. While SysML variant modeling provides the opportunity to identify some of the components of a system architecture as decisions, the formulations allowed by SysML variants are extremely restricted, especially in the type of decisions (Combining decisions only). This could be remedied for example through the development a new SysML stereotype for a decision with which block definition diagrams and activity diagrams can be defined. Such view of the system could then be incorporated into an architecture framework as a decision space viewpoint, which is of interest to some stakeholders.

In the application of the patterns as quantitative prescriptive decision support, the development of a library containing reusable pattern-specific knowledge (e.g. algorithms to enumerate and count architecture fragments from the different patterns, efficient operators to do local search on different patterns) is left for future work. One of the major challenges is in the combination of pattern-specific knowledge in arbitrary problems. While this combination is trivial in the case of independent decisions, and it is relatively simple in some cases, it could become difficult for problems with complex dependencies between decisions. Furthermore, if new algorithms are to be developed for each of the patterns, it is necessary to have a means for comparing the performance of the new algorithms with respect to the existing ones over a wide range of problems that are relevant to system architecture. This suggests the development of a set of benchmarking problems that span the six patterns introduced in this paper and have different degrees of complexity as well as different features in their tradespaces. Indeed, existing sets of benchmarking problems in multi-objective optimization or combinatorial optimization may not be appropriate since they are too different from real-life architecture problems. The main challenge is perhaps the development of value functions to use in these benchmarking problems. One possible research direction for this is to use the VASSAR method for developing value functions and generate different value functions by simply generating arbitrary sets of stakeholder requirements, component capabilities, and emergence rules [65].

References

[1] E. Crawley, B. Cameron, and D. Selva, *Systems Architecture: Strategy and Product Development for Complex Systems*. Prentice Hall, 2015.

[2] T. Weilkiens, *Systems engineering with SysML/UML: modeling, analysis, design*. Heidelberg, Germany: The Morgan Kaufmann/OMG Press, 2006.

[3] D. Dori, *Object-Process Methodology: A holistic paradigm*. Berlin, Heidelberg: Springer, 2002.

[4] C. S. L. N. I. of S. and Technology, "INTEGRATION DEFINITION FOR FUNCTION MODELING (IDEF0)," *Draft Federal Information Processing Standards Publication 183*. pp. 1–128, 1993.

[5] M. W. Maier and E. Rechtin, *The Art of Systems Architecting*. New York: CRC press, 2000.

[6] E. Rechtin, *System Architecting, Creating & Building Complex Systems*. Englewood Cliffs, NJ: Prentice Hall, 1991.

[7] C. Alexander, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1994.

[9] D. V. Steward, "The design structure system: A method for managing the design of complex systems," *IEEE Trans. Eng. Manag.*, vol. 28, pp. 71–74, 1981.

[10] S. D. Eppinger and T. R. Browning, *Design Structure Matrix Methods and Applications*. The MIT Press, 2012.

[11] K. Kalligeros, O. De Weck, and R. De Neufville, "Platform identification using Design Structure Matrices," in *Sixteenth Annual International Symposium of the International Council On Systems Engineering (INCOSE)*, 2006.

[12] G. J. Klir, *Approach to general systems theory*. New York, NY: Van Nostrand Reinhold, 1969.

[13] F. Harary, R. Z. Norman, and D. Cartwright, *Structural models: An introduction to the theory of directed graphs*. {John Wiley & Sons Inc}, 1965.

[14] J. N. Warfield, *Societal systems: Planning, policy, and complexity*. World Scientific, 1976.

[15] A. D. Hall, *Metasystems methodology: a new synthesis and unification*. Pergamon, 1989.

[16] S. Pugh, *Total design: integrated methods for successful product engineering*. Workingham: Addison Wesley Publishing Company, 1991.

[17] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *Eur. J. Oper. Res.*, vol. 48, no. 1, pp. 9–26, Sep. 1990.

[18] R. a. Howard and J. E. Matheson, "Influence Diagrams," *Decis. Anal.*, vol. 2, no. 3, pp. 127–143, 2005.

[19] G. A. Hazelrigg, "Letter to the Editor re:'The Pugh controlled convergence method: model-based evaluation and implications for design theory,'" *Res. Eng. Des.*, vol. 21, no. 3, pp. 143–144, 2010.

[20] H. A. Simon, "Theories of bounded rationality," *Decis. Organ.*, vol. 1, no. 1, pp. 161–176, 1972.

[21] G. A. Hazelrigg, "A Framework for Decision-Based Engineering Design," *J. Mech. Des.*, vol. 120, no. 4, p. 653, 1998.

[22] P. Y. Papalambros and D. J. Wilde, *Principles of Optimal Design*. Cambridge University Press, 2000.

[23] J. Sobieszczanski-Sobieski, *Multidisciplinary design optimization: an emerging new engineering discipline*. Springer, 1995.

[24] R. Wang and C. H. Dagli, "Executable system architecting using systems modeling language in conjunction with colored Petri nets in a model-driven systems development process," *Syst. Eng.*, vol. 14, no. 4, pp. 383–409, 2011.

[25] W. L. Simmons, "A Framework for Decision Support in Systems Architecting," PhD dissertation, Massachusetts Institute of Technology, ProQuest/UMI, Ann Arbor, 2008.

[26] H. B. Koo, E. F. Crawley, and C. Magee, "A Meta-language for Systems Architecting," Massachusetts Institute of Technology, 2005.

[27] D. E. Knuth, *The Art of Computer Programming Volume 4A Combinatorial Algorithms Part 1*. Pearson, 2011.

[28] A. Rudat, "Tradespace Exploration Approach for Architectural Definition of In-space Transportation Infrastructure Systems for Future Human Space Exploration," in *International Astronautical Congress, 63rd*, 2012, pp. 1–15.

[29] K. T. Ulrich and S. D. Eppinger, *Product Design and Development*, vol. 384. 1995.

[30] International Organization Of Standardization, "ISO/IEC/IEEE 42010:2011 - Systems and software engineering -- Architecture description," 2011.

[31] C. Dickerson and D. N. Mavris, *Architecture and Principles of Systems Engineering*. Auerbach Publications, 2009.

[32] D. M. Buede, *The Engineering Design of Systems: Models and Methods*. Wiley, 2009.

[33] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12. pp. 1053–1058, 1972.

[34] N. Suh, "Axiomatic design theory for systems," *Res. Eng. Des.*, vol. 10, no. 4, pp. 189–209, 1998.

[35] K. Holtta-Otto and O. de Weck, "Degree of Modularity in Engineering Systems and Products with Technical and Business Constraints," *Concurr. Eng.*, vol. 15, no. 2, pp. 113–126, 2007.

[36] J. N. Warfield, "Structuring complex systems," Battelle, Office of Corporate Communications, BOOK 4, 1978.

[37] J. N. Warfield, "An assault on complexity," Battelle, Office of Corporate Communications, BOOK 3, 1973.

[38] J. N. Warfield and J. D. Hill, "A unified systems engineering concept," Battelle, Office of Corporate Communications, BOOK 1, 1972.

[39] W. L. Chapman, J. Rozenblit, and a. T. Bahill, "System Design is an NP-Complete Problem," *Syst. Eng.*, vol. 4, no. 3, pp. 222–229, 2001.

[40] D. Selva, B. G. Cameron, and E. F. Crawley, "Rule-Based System Architecting of Earth Observing Systems: Earth Science Decadal Survey," *J. Spacecr. Rockets*, vol. 51, no. 5, pp. 1505–1521, 2014.

[41] A. A. Golkar, R. Keller, B. Robinson, O. L. de Weck, and E. F. Crawley, "A methodology for system architecting of offshore oil production systems," in *Proceedings of the international design structure matrix conference (DSM'09). Greenville, South Carolina, USA*, 2009, pp. 343–356.

[42] A. Dominguez-Garcia, G. Hanuschak, S. Hall, and E. Crawley, "A Comparison of GN&C Architectural Approaches for Robotic and Human-Rated Spacecraft," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2007, pp. 20–23.

[43] M. Sanchez-Net, I. Del Portillo, B. G. Cameron, E. F. Crawley, and D. Selva, "Integrated Tradespace Analysis of Space Network Architectures," *J. Aerosp. Inf. Syst.*, vol. 12, no. 8, pp. 564–578, 2015.

[44] R. Patel, W. Paleari, and D. Selva, "Architecture Study of an Energy Microgrid," in *IEEE Systems of Systems Conference (SoSE)*, 2016, pp. 1–8.

[45] M. Ehrgott and X. Gandibleux, "A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization," *OR Spectr.*, vol. 22, no. 4, pp. 425–460, Nov. 2000.

[46] D. C. Montgomery, *Design and Analysis of Experiments*, 8th ed. New York: Wiley, 2010.

[47] F. Zwicky, *Discovery, Invention, Research through the morphological approach*. Macmillan Publishing Company, 1969.

[48] D. Selva, B. G. Cameron, and E. F. Crawley, "Rule-based System Architecting of Earth Observing Systems: The Earth Science Decadal Survey," *J. Spacecr. Rockets*, 2014.

[49] H. Mühlenbein, "Parallel Genetic Algorithms, Population Genetics and Combinatorial

Optimization," *Proc. third Int. Conf. Genet. Algorithms*, pp. 398–406, 1989.

[50] N. J. Radcliffe, "Equivalence Class Analysis of Genetic Algorithms," *Complex Syst.*, vol. 5, no. 2, pp. 183–205, 1991.

[51] T. R. Browning, "Applying the Design Structure Matrix to System Decomposition and Integration Problems : A Review and New Directions," *IEEE Trans. Eng. Manag.*, vol. 48, no. 3, pp. 292–306, 2001.

[52] S. Sahni, "Approximate Algorithms for the 0/1 Knapsack Problem," *J. ACM*, vol. 22, no. 1, pp. 115–124, 1975.

[53] W. Wymore, *Model-based Systems Engineering*. CRC Press, 1993.

[54] T. Browning and S. Eppinger, "Modeling impacts of process architecture on cost and schedule risk in product development," *Eng. Manag. IEEE …*, vol. 49, no. 4, pp. 428–442, 2002.

[55] G. Dantzig and R. Fulkerson, "Solution of a large-scale traveling-salesman problem," *J. Oper. Res. Soc.*, vol. 1934, 1954.

[56] A. S. Manne, "On the Job-Shop Scheduling Problem," *Oper. Res.*, vol. 8, no. 2, pp. 219–223, 1960.

[57] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.

[58] J. Hartmanis, "Computers and Intractability: A Guide to the Theory of NP-Completeness (Michael R. Garey and David S. Johnson)," *SIAM Review*, vol. 24, no. 1. pp. 90–91, 1982.

[59] N. Kano, N. Seraku, F. Takahashi, and S. Tsuji, "Attractive quality and must-be quality," 1984.

[60] D. E. Knuth, *The Art of Computer Programming Volume 2 Seminumerical algorithms*. Pearson, 2011.

[61] A. J. Stam, "Generation of a random partition of a finite set by an urn model," *J. Comb. Theory, Ser. A*, vol. 35, no. 2, pp. 231–240, 1983.

[62] D. E. Goldberg and R. jun. Lingle, "Alleles, loci and the Traveling Salesman Problem," in *Genetic algorithms and their applications, Proc. 1st Int. Conf.*, 1988, pp. 154–159.

[63] L. Denœud-Belgacem, "Transfer distance between partitions," *Adv. Data Anal. Classif.*, vol. 2, pp. 279–294, 2008.

[64] L. Vygotsky, "Interaction between learning and development," in *Mind and Society*, Cambridge, MA: Harvard University Press, 1978, pp. 79–91.

[65] D. Selva and E. Crawley, "VASSAR: Value Assessment of System Architectures using Rules," in *Aerospace Conference, 2013 IEEE*, 2013.

[66] R. M. Ross, G. T.; Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem," *Math. Program.*, no. 8, pp. 91–103, 1975.

[67] R. K. Ahuja, a. Kumar, K. C. Jha, and J. B. Orlin, "Exact and Heuristic Algorithms for the Weapon-Target Assignment Problem," *Oper. Res.*, vol. 55, no. 6, pp. 1136–1146, 2007.

[68] A. Drexl, "A simulated annealing approach to the multiconstraint zero-one knapsack problem," *Computing*, vol. 40, no. 1, pp. 1–8, Dec. 1988.

[69] R. S. Garfinkel and G. L. Nemhauser, "The Set-Partitioning Problem: Set Covering with Equality Constraints," *Oper. Res.*, vol. 17, no. 5, pp. 848–856, Sep. 1969.

[70] S. E. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," *Oper. Res.*, vol. 17, no. 3, pp. 395–412, 1969.

[71] J. Edmonds, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," *Computing*, vol. 19, no. 2, pp. 248–264, 1972.

[72] R. Graham, “On the history of the minimum spanning tree problem,” *Ann. Hist. Comput.*, vol. 7, no. 1, pp. 43–57, 1985.

[73] P. Hansen and B. Jaumard, “Algorithms for the maximum satisfiability problem,” *Computing*, vol. 303, pp. 279–303, 1990.

Appendix: Classical combinatorial optimization problems

We present a list of some classical combinatorial optimization problems that are relevant for systems architecture. For each problem, a reference is provided. Many of these problems have several hundred years, and were first introduced in mathematical contests or challenges of the time (e.g., Sir Hamilton's Icosian game introducing the traveling salesman problem in the early 1800's). Thus, the reference provided is always representative of the foundational work on methods to solve the problem, but not necessarily the first time the problem was introduced.

Assignment problem and variants [66]: In the original assignment problem, there are a number of agents and a number of tasks, and each agent has a different cost to realize each task. The goal of the problem is to assign each task to exactly one agent, in such a way that all tasks are performed, and the total cost is minimized. In a generalized version of the assignment problem, agents can perform more than one task, and they have a given budget that cannot be exceeded. Furthermore, when performing a task, each agent produces a certain profit, and the goal of the problem is thus to maximize profit subject to not exceeding any agent budget. However, each task must be assigned to exactly one agent. In the weapon-target assignment problem [67], the latter condition is relaxed, so that a task (target) can be assigned to any number of weapons (agents) including all or none.

Travelling salesman problem and variants [55]**:** In the traveling salesman problem, there is a list of cities and a matrix containing their pairwise distances. The goal of the problem is to find the shortest path that passes through each city exactly once and returns to the city of origin. The vehicle routing problem is a generalization where there is a fleet of vehicles and a set of customer locations to deliver.

Knapsack problem and variants [68]: In the original knapsack problem, there is a list of items, each with a certain value and a certain weight. The goal of the problem is to determine the optimal number of items of each type to choose in order to maximize value for a certain maximum cost. In the 0/1 version of the same problem, the number of each item can only take the values {0,1}. In the multiple knapsack problem, there are multiple knapsacks, and the subsets of items chosen for each knapsack must be disjoint.

Set partitioning and set covering problems [69]: In the original set covering problem, there is a list of elements referred to as the universe, and a number of predetermined sets of elements, whose union is the universe. The goal of the set covering problem is to identify the minimum number of these predetermined subsets for which the union is still the universe. The set partitioning problem is a constrained version of the set covering problem where the selected subsets need to be disjoint or mutually exclusive. In other words, each element can only appear in one set in the set partitioning problem, while it may appear in more than one set in the set covering problem.

Job-shop scheduling and variants [56]: In the simplest version of the job-shop scheduling problem, there is a list of jobs that needs to be assigned to a set of available resources (e.g., machines). Each job has a certain duration on each machine. The goal of the problem is to find the sequence of assignments of jobs to machines that minimizes the combined duration of the tasks. Variations of the problem include for example constraints between tasks (e.g., a task needs to occur before another task), costs for running the machines, or constraints between machines (e.g., two machines cannot be running simultaneously).

Shortest path problem [70]: In a shortest path problem, there is a graph defined by a list of nodes and a list of edges, and each edge has a distance associated to it. The goal of the problem is to find the path between two given nodes that minimizes the total distance.

Max flow problem and variants [71]: In the original formulation of the max flow problem, there is a graph with a single source node, a single sink node and a number of other nodes. Edges connecting those nodes are capacitated, and the goal is to find a feasible path from the source to the sink that minimizes cost. In the most generic formulation of a network flow problem, there is a graph defined by a list of nodes and a list

of edges. Nodes can be sources or sinks. Sources have a positive flow (supply) while sinks have a negative flow (demand). Each edge has a capacity associated to it, so that the flow through that edge cannot exceed that capacity. Each edge also has a cost associated to it. The goal of the network flow problem is to transport all the flow from the supply nodes to the demand nodes at minimum cost.

Minimum spanning tree [72]: In the minimum spanning tree problem, there is a graph defined by a list of nodes and a list of edges. Each edge has a cost associated to it. A spanning tree is defined as a subset of the edges that form a tree that contains all nodes. The goal of the problem is to find the spanning tree of minimum cost.

Maximum satisfiability (MAX-SAT) problem [73]: In the maximum satisfiability problem, there is a set of Boolean variables, and a list of logical clauses that use that set of variables. The goal of the problem is to find the assignment to that set of Boolean variables that maximizes the number of clauses that are satisfied.