Combinatorial Models for Heterogeneous System Composition and Analysis

Saigopal Nelaturi
and Johan de Kleer
System Sciences Laboratory
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304
Email: nelaturi, dekleer@parc.com

Vadim Shapiro
Spatial Automation Laboratory
1513 University Avenue
University of Wisconsin, Madison
Madison, Wisconsin - 53703
Email:vshapiro@engr.wisc.edu

***Abstract*—Complex systems with high dimensional design spaces are often specified by subsystem compositions that constrain feasible designs towards tight regimes in the design space. Composition between (sub)systems is limited by the interfaces exposed by the interacting architectures. Treating architectures as first class objects with their own formal properties is required to enable designing complex systems by novel compositions of heterogeneous subsystems and their design spaces. We describe a combinatorial model for system architectures using the formalism of *cellular sheaves* to identify all feasible compositions across heterogeneous subsystems through exposed interfaces. We discuss how the proposed model may be used to automatically generate novel systems of systems compositions and test key properties of composability and compositionality, required for solving planning and reconfiguration problems in systems of systems design.**

I. INTRODUCTION

Most complex physical systems (e.g. cyber-physical systems) share the main characteristics of embedded systems [1]. Such systems are *platform-aware* implying that the system design depends on the execution platform (e.g. system on a chip, distributed system) as much as spatio-temporal physical constraints. In other words, the system design space is constrained simultaneously by the end application and by the particular platform architecture used to implement a design. Although designing platform-aware complex systems is often a substantially hard problem (e.g. designing aerospace or robotic surgery systems), the conceptual framework pushes system designs towards solutions operating efficiently within tight regimes in the combined design space of all interacting subsystems. Modularity is the preferred outcome but not a key performance criterion of such designs.

Recognizing that modularity and reuse leads to greater system adaptivity has driven the development of flexible manufacturing and robotic systems, which is based on the idea of building reconfigurable production systems from standardized modular subsystems [2]. Such systems are still platform based, albeit with an increased feasible design space due to reconfigurability. Despite the size and complexity of platform-based systems, their design spaces are possibly hard to precisely characterize but always known a-priori and are often fixed to match application specific goals.

The approach of working with a-priori known design spaces helps component and small subsystem design but eventually leads to two substantial problems. First, system integration becomes extremely challenging because interfaces are defined a-posteriori. Essential design concerns are usually separated into physical systems, software, and platform engineering; poorly understood interactions and conflicts across design domains appear at the system integration stage [3]. Second, the component-based approach depends on key properties of *compositionality* (system level properties can be computed from component properties) and *composability* (component properties do not change by virtue of interactions with other components) [4]. Compositionality and composability must not adversely affect important system and component properties such as safety, stability, and performance. Therefore system integration often requires extensive analysis and simulation.

Establishing compositionality for physical system properties (such as stability) is non-trivial even for simple cases. For example two objects (e.g. pencils on a table) may be in stable equilibrium and may be composed via compatible interfaces (contact surfaces) by placing one object over another. However, the resulting composition is in (possibly unstable) equilibrium only for a very small set of relative configurations obtained by balancing one object over another. Verifying compositionality requires modeling and simulation to find parameters where properties of composed systems may be derived from properties of system components. Similarly, establishing composability may be fairly easy for component based systems but is not so for general combinations; for example chemical reactions fundamentally alter the structure of entities (elements become compounds) after composition and the physical properties of these entities are lost (e.g. oxygen supports combustion but water does not). Therefore model based simulation is critical to evaluate properties of composed systems. However, compositionality or composability can only be evaluated after explicitly checking that interface *types* are compatible.

In this paper we consider a *dual* design paradigm where the primary objects are not component design/state spaces but

rather the exposed interfaces for composition. The architecture and design spaces for composition are *not* known a-priori, resulting in a platform-agnostic environment. However, the available interfaces are pre-defined so that interoperability of components and architectures is enabled by construction. This paradigm anticipates a scenario where services derived from networked cyber-physical and software systems are dynamically deployed and composed to enable multi-functional devices and systems. For example, one could envision a medical emergency scenario where a cyber-physical system uses location based services to guide the vehicle carrying a patient to the nearest hospital, informing the hospital in advance of the patient's arrival and medical condition while notifying the patient's family through a mobile device [5].

Designing composed heterogeneous systems of systems requires rapid exploration, planning, and mapping across software and cyber-physical services. We claim that the ability to rapidly generate and test design *spaces* (for compositionality and composability) is afundamental requirement to achieve such composition. Traditional design space exploration attempts to find novel designs in a fixed design space, whereas the interface driven approach aims to find novel compositions of design spaces that are capable of solving unanticipated applications. The interface-centric design space exploration is dual to traditional design space exploration and is better equipped to solve problems whose solutions require reconfiguration of existing and deployed systems, as opposed to synthesizing designs for components and subsystems that combine to form new systems. It is critical that each such composition is accompanied with a model based simulation that evaluates compositionality and composability.

A-priori defining composable interfaces combinatorially describes the space of architectures generated by all available and feasible interface compositions. In this paper we describe how feasible architectures have the topological structure of a *cell-complex* generated through combinatorial interactions of interfaces. Associating data available at the interfaces to each element in the cell-complex represents the architecture as a *cellular sheaf*. The cellular sheaf representation of system architectures implicitly encodes all local combinations of feasible and compatible interface compositions that may be assembled into a global system composition, so that for each composition a model based simulation can evaluate compositionality and composability. Each global composition (i.e. interface combination) corresponds to a design space over which the simulation is posed.

II. Related Work

A. Service oriented computation

System composition through interfaces is related in spirit to the paradigm of service oriented computing. Services (e.g. web services) are self-describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications [13] (e.g. over the internet) using a standardized communication protocol (e.g. SOAP). Applications are developed by composing services with well-defined interfaces, while abstracting away internal details. While this approach works well in particular for web services, interface compatibility is a necessary but insufficient condition for compositionality of cyber-physical systems. For example, the (compositional) stability of a control system implemented with networked controllers is highly dependent on nonlinear effects of probabilistic delays, uncertain processor availability, and signal quantization [3]. The ability to accurately model and simulate particular composition architectures is key to evaluating the efficacy of designs generated by composing autonomous systems treated as services.

The service oriented computing paradigm is inherently modular, allowing for service (or system) reuse and extension by treating a set of services (or systems) as a programming library. Conceptually, service composition is predicated on the existence of standardized reference semantics and protocols that express how services in a library expose their behavior through an API. Efforts in automated service composition [14], [15], [16] have shown that the construction of an orchestrated service as a finite state transition machine (transitioning between inputs and outputs of the composed services) is EXPTIME-hard in the presence of temporal constraints. Essentially the behavior of each system is abstracted into a set of (internal) states, actions, and transitions, and the composition of such behaviors may be accomplished in time exponential in the number n of behaviors. Therefore it is important that n is kept small so that composition algorithms are tractable. Systems in the library are composable (see Section I) because behaviors of individual systems are preserved in the orchestrated service.

The key requirements for effective system design through composition are

- The ability to specify systems through composable interfaces that abstract away internal implementations and platform dependencies
- A description of composed interface architecture
- The ability to systematically generate and test multiple architectures
- A model based simulation that evaluates compositionality

B. Combinatorial system design and analysis

Physical systems are usually modeled, designed, and analyzed using a differential (i.e. smooth) comprehension of the world, but the models and analysis are always implemented using discrete representations and algorithms. Many of these data structures, such as those used in finite element methods, are combinatorial because the domain on which the physical problem is posed may be approximated by a finite combination of primitive elements. The accuracy of the physical analysis is therefore only as good as the accuracy of the approximation. However, using tools of discrete exterior calculus [9], it has been shown that the finite data structure itself can be used to extract physically meaningful conservation laws [10], [11] without requiring the differential formulation. Essentially, the designed object is represented as an oriented cell complex [9] and physical quantities such as current, mass, force etc can be formally defined as algebraic objects called chains and cochains. The immediate consequence of this representation is

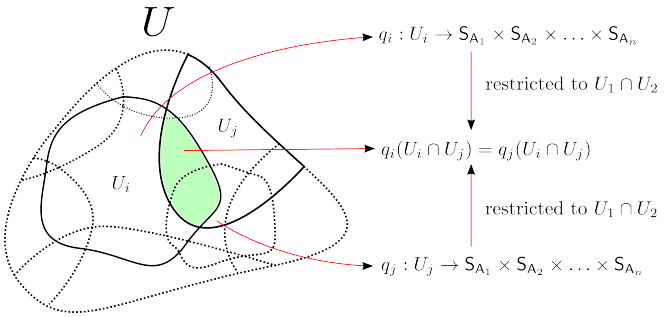


Fig. 1. The sheaf condition highlighted for sets U_i and U_j in a collection of sets whose union is U. At every set U_i in the collection, a query is defined such that the range of the query is a design space $\mathsf{S}_{\mathsf{A}_1} \times \mathsf{S}_{\mathsf{A}_2} \times \ldots \times \mathsf{S}_{\mathsf{A}_n}$. If queries q_i and q_j (with common design space) defined over sets U_i and U_j agree at the intersection $U_i \cap U_j$ for every pair U_i, U_j of sets in the collection, then there is a unique query $q : U \to \mathsf{S}_{\mathsf{A}_1} \times \mathsf{S}_{\mathsf{A}_2} \times \ldots \times \mathsf{S}_{\mathsf{A}_n}$ such that the restriction of q to U_i is q_i.

that most spatial, physical, and design laws may be expressed algebraically and applied on the cell complex.

We first show (in Section III) necessary conditions for composability through interface composition, and then show (in Section IV) that interface combinations have the structure of a cell complex. Using the observations made on service oriented computing and combinatorial design, we propose cellular sheaves as a data structure that encapsulates necessary conditions for interface composability and compositionality. Elementary operations on cell complexes then allow generating new interface architectures that can be used to test compositionality over alternative system compositions.

III. Interfaces in System Composition

A. System models as sheaves

We will build upon the framework of categorical systems theory [12], where objects/systems in the real world are modeled in terms of a set of maps (which we call queries) into design spaces, such that the maps describe observable system properties. We model systems living in $U = X \times T$ of $\mathbb{R}^3 \times (\mathbb{R}_+ \cup 0)$ (three spatial dimensions and one temporal dimension with a distinguished initial time 0). Every system S in U is expected to have an interface at which S's *attribute spaces* $\mathsf{S}_{\mathsf{A}_1}, \ldots, \mathsf{S}_{\mathsf{A}_n}$ are queried through a combination of sensors and updated through in-built models. A *query* q is a map from U to a product space of finitely many attribute spaces. Each such product space is considered as a design space where particular attribute spaces are queried. In other words, the *range of a query is a design space*. Notice that a system may have multiple design spaces so that application specific queries may be constructed into the relevant design space. The image $q(u)$ of a query evaluated at $u \in U$ onto a design space returns an instance of the system attributes that describe the system in a physically meaningful way; i.e. queries are the fundamental entities that associate physical meaning to a system.

When the *sheaf condition* [12] is satisfied (see Figure 1), local queries q_i defined over subdomains $U_i \subset U$ can be fused into a *unique* global system query q over $U = \cup U_i$ whose restriction to each U_i returns the corresponding q_i. In other words, the sheaf condition says that any set of pairwise consistent queries (i.e. queries that agree where their domains overlap) can be fused together into a unique query over the union of their domains. This is very useful, because it allows recursively building queries over large U by modeling them piecewise in smaller U_i. In reality a set Q_i of queries may be posed at each U_i. Then the sheaf Q is the set of queries defined over U which includes all q that meet the sheaf condition and describe the system's relevant states/events/physical properties. Q is called the *system model*. Queries in the system model may be related through maps between design spaces called *constraints*. The system interface is the set $\mathsf{Q}(U)$ of images $q(U)$ for each query $q \in \mathsf{Q}$ (constrained by other queries) in the system model.

Goguen [12] provides many examples of concurrent interacting systems modeled as sheaves. A particularly illustrative example is when the system model is a sheaf that models a deterministic (but not necessarily finite state) machine S. The attribute spaces are $\mathsf{S}_{\mathsf{A}_1}$ (inputs), $\mathsf{S}_{\mathsf{A}_2}$ (states), and $\mathsf{S}_{\mathsf{A}_3}$ (outputs). Suppose we are given an initial state σ , a state transition function $\delta : \mathsf{S}_{\mathsf{A}_1} \times \mathsf{S}_{\mathsf{A}_2} \to \mathsf{S}_{\mathsf{A}_2}$, an output function $\beta : \mathsf{S}_{\mathsf{A}_2} \to \mathsf{S}_{\mathsf{A}_3}$, and a system model $\mathsf{Q} = \{q_1, q_2, q_3\}$. The queries in Q are defined as follows: $q_1 : U \to \mathsf{S}_{\mathsf{A}_1}$ returns an input, $q_2 : U \to \mathsf{S}_{\mathsf{A}_2}$ returns a state, and $q_3 : U \to \mathsf{S}_{\mathsf{A}_3}$ returns an output. The design spaces in the system model are the ranges of the queries, i.e. $\mathsf{S}_{\mathsf{A}_1}, \mathsf{S}_{\mathsf{A}_2}, \mathsf{S}_{\mathsf{A}_3}$. The queries can be thought of as signals that respect the sheaf condition; for example the input query q_1 may be defined 'piecewise' so that the pieces agree only at the overlaps between the $U_i \subset U$. It is usually convenient to model discrete time. Then Q for the deterministic machine/automata has the following constraints

$$q_2(x,0) = \sigma \tag{1}$$

$$q_3(x,t) = \beta(q_2(x,t)) \tag{2}$$

$$q_2(x,t+1) = \delta(q_1(x,t+1), q_2(x,t)) \tag{3}$$

It can be checked that q_2, q_3 also meet the sheaf condition. Note that non-determinstic automata can be modeled by redefining $\delta : \mathsf{S}_{\mathsf{A}_1} \times \mathsf{S}_{\mathsf{A}_2} \to 2^{\mathsf{S}_{\mathsf{A}_2}}$ to return a set of states, and updating the constraint in Equation 3 as $q_2(x,t+1) \in \delta(q_1(x,t+1), q_2(x,t))$.

In many cases it may be hard to explicitly define the system model Q, for example to characterize a complex system such as an NC machine tool. However, it is possible to *test* the sheaf condition via constraints on queries that monitor important attributes such as spindle speeds and power consumption. Unexpected or faulty behavior of physical systems occurs when the sheaf condition is lost. Therefore, we assume every query in a system model Q satisfies the sheaf condition.

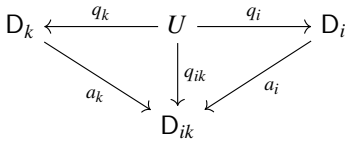
B. Composability

The sheaf condition implies composability of queries because local queries are preserved when q is restricted to the U_i (see Figure 1). However, the sheaf condition by itself is a weak statement of composability because it relies exclusively on restriction maps at intersecting subdomains (see Figure 1). Queries from distinct but composable systems may not

restrict identically at the intersections of the U_i to produce a sheaf without some non-trivial transformation, e.g. when components are queried in independent coordinate systems but have to be merged into (and queried in) a common coordinate system for assembly.

Let us assume there are multiple systems that can interact in U. For systems $\mathsf{S}^i, \mathsf{S}^k$ to be composable, interfaces to the services they provide must be interoperable. To formalize this notion, consider queries $q_i : U \to \mathsf{D}_i, q_k : U \to \mathsf{D}_k$ in the system models of $\mathsf{S}^i, \mathsf{S}^k$. The systems are composable if and only if there exist *attachment* maps a_i and a_k that map the design spaces $\mathsf{D}_i, \mathsf{D}_k$ into a common design space D_{ik} such that

- $a_i \circ q_i = a_k \circ q_k$. Here $\circ$ indicates usual function composition. Equivalently we may say the diagram in Equation 4 commutes (all sequences of arrows initiating at U and terminating at D_{ik} are equivalent).



(4)

  Diagram commutativity implies that the images of the queries match after attachment into the common design space. Notice that an interface is defined through maps from U into a design space. Therefore diagram commutativity implies an interface through the query $q_{ik} : U \to \mathsf{D}_{ik}$ for the composed system S^{ik}. It may be shown that q_{ik} satisfies the sheaf condition [12].
- There are maps $(a_i)^{-1} : \mathsf{D}_{ik} \to \mathsf{D}_i$ and $(a_k)^{-1} : \mathsf{D}_{ik} \to \mathsf{D}_k$ such that $a_i \circ (a_i)^{-1}$ and $a_k \circ (a_k)^{-1}$ are identity maps on D_{ik}. If such a condition does not exist, then the individual systems $\mathsf{S}^i, \mathsf{S}^j$ cannot co-exist with their composition and are therefore not composable. For example elements and their compounds cannot simultaneously co-exist in a chemical composition, therefore they cannot be simultaneously queried.

Systems S^i and S^k with models Q_i and Q_k are composable if and only if there exist $q_i \in \mathsf{Q}_i$ and $q_k \in \mathsf{Q}_k$ that satisfy the above properties. The composed system model Q_{ik} contains all queries q_{ik} for which there exist commutative diagrams such as the one shown in Equation 4. The first property ensures that the constituent systems being composed have compatible interface types, otherwise composition is not possible. The second property ensures that the constituent systems co-exist with the composition, and can still be queried to check that their individual properties preserved after composition. When these properties are satisfied, the resulting $\mathsf{Q}_{ik}(U)$ is the *composed system interface*.

In general, the image of any $(a_i)^{-1}$ should be a subset of $q_i(U)$ to indicate that the original interfaces are preserved but constrained by the composition. In non-obvious cases, characterizing the $(a_i)^{-1}$ and $(a_k)^{-1}$ may require solving an inverse problem where $q_{ik}(U), a_i, a_k$ are known but the constituent $q_i(U)$ and/or $q_k(U)$ (after composition) are not. Composability is then determined by solving for $q_i(U), q_k(U)$ and checking if they are operating as designed and that $a_i \circ (a_i)^{-1}$ and $a_k \circ (a_k)^{-1}$ are identity maps on D_{ik}. Alternatively, given a q_i and a_i, a forward problem could be solved with given attachment maps to explicitly identify the subset of $q_i(U)$ for which composition is feasible. Then the $(a_i)^{-1}$ will map onto this subset.

Since all maps originate from a common source U, and interfaces are defined through commutative diagrams, we will use

$$\mathsf{D}_k \to \mathsf{D}_{ik} \leftarrow \mathsf{D}_i \tag{5}$$

as an equivalent compressed notation for the commutative diagram in Equation 4. When the properties for composability are met, more complex diagrams such as the one shown in Figure 2 may be constructed when multiple composable interfaces interact in a domain U.

IV. Combinatorial Description of Interoperable System Models

A. Data structures for system composition

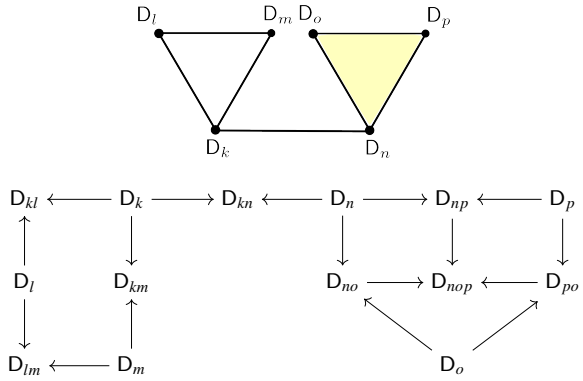


Fig. 2. A simplicial complex representing composable interactions between design spaces $\mathsf{D}_k \ldots \mathsf{D}_p$. Observe the distinction between the compositions corresponding to design spaces $\mathsf{D}_n, \mathsf{D}_o, \mathsf{D}_p$ (connected by a triangle) and $\mathsf{D}_k, \mathsf{D}_l, \mathsf{D}_m$ (connected by 3 edges). $\mathsf{D}_k, \mathsf{D}_l, \mathsf{D}_m$ can only communicate pairwise but $\mathsf{D}_n, \mathsf{D}_o, \mathsf{D}_p$ share a common interface.

Given a collection of potentially interoperable systems modeled in a domain U, we first observe that all possible interactions can be captured in the form of an abstract *simplicial complex*. A simplicial complex is combinatorial description of a topological space by gluing together topological primitives called simplices. Simplices can also be given geometric meaning; a $k-$simplex is the convex hull of $k+1$ affinely independent points. We use special names for the first few dimensions, a 0-simplex is called a vertex, a 1-simplex is called an edge, a 2-simplex is called a triangle, and a 3-simplex is called a tetrahedron.

Every design space of every system S^i in U is represented as a vertex. When a pair of systems $\mathsf{S}^i, \mathsf{S}^k$ are composable through design spaces D_i and D_k, an undirected edge may be drawn between the design spaces to represent a composable interface (i.e. $\mathsf{D}_k \to \mathsf{D}_{ik} \leftarrow \mathsf{D}_i$ is well defined). When three systems are composable, the corresponding composed systems interface is represented as a 2-simplex, represented geometrically as a triangle with interior to indicate all possible interactions between the vertices (see Figure 2).

In a commutative diagram each query q_i composed with an attachment map a_i satisfies the sheaf condition. Each such composition $a_i \circ q_i$ is an element of the composed system model. When all the design spaces have the same type (e.g. if all the design spaces are vector spaces), the composed system model is a sheaf of that type (e.g. vector spaces) over a simplicial complex [6]. The attachment maps are defined between design spaces, and so can be thought of as constraints in the composed system model (see Section III-A). Notice that we have progressed from defining query composition over subdomains $U_i \subset U$ through the sheaf condition, to establishing composability via interface attachments in Section III, and now we see that composability again leads to a type of sheaf, albeit defined over a combinatorial space. Each composed interface is represented geometrically as a $k-$dimensional simplex where k indicates the number of interacting systems in the composition.

Sheaves over simplicial complexes, also called *cellular sheaves*, are data structures that may be used to represent the interaction between sensors (sources of data) and their data fields in a mutually consistent manner [7]. The correspondence between cellular sheaves and composable systems implies a dynamic cellular sheaf within a domain U may be maintained to describe interoperability of systems living in U. When new systems appear in U, their design spaces may be added or *glued* to existing simplicial complexes after checking the newly available interfaces are composable. Similarly, when systems are removed from U, the simplicial complex is updated through an operation called a collapse [8]. Informally, removing a design space D_i deletes all higher dimensional simplices[1] that contain D_i. The system sheaf is automatically updated when the simplicial complex is modified.

B. Generating and Testing Design Spaces

Recall that establishing composability may require solving an inverse problem in complex cases (see Section III-B) which can be challenging if the interface compositions constrain the constituent system design spaces. Once composability is established and the system sheaf is constructed, combinations of interacting interfaces can be generated systematically by enumerating topologies through gluing and collapsing operations. Each such combination requires solving a forward problem to test compositionality, which again could be challenging depending on the queries in the system model [3]. However, we must require the existence of a model based simulation if compositionality is computationally evaluated.

We now describe a simple example that illustrates systematically generating and testing design spaces arising from system compositions. Consider six systems $\mathsf{S}^1,\ldots,\mathsf{S}^6$ with models $\mathsf{Q}_1,\ldots,\mathsf{Q}_6$. For simplicity, let us assume each Q_i has a single query q_i and that every design space is a vector space. Each system S^i therefore consists of a single interface $\mathsf{Q}_i(U) = \{q_i(U)\}$ at which the results of internal computation at $u \equiv (x,y,z,t) \in U$ return the following vectors

$$\begin{aligned}
q_1(u) &= (Q,A,B,1 \ if \ t<10 \ else \ 0)\\
q_2(u) &= (C,D,E,1 \ if \ \sqrt{x^2+y^2+z^2}<25 \ else \ 0)\\
q_3(u) &= (P,F,D,1 \ if \ z>10 \ else \ 0)\\
q_4(u) &= (B,G,Q,1)\\
q_5(u) &= (C,D,1)\\
q_6(u) &= (C,H,Q,1)
\end{aligned}$$

Each vector $q_i(u)$ represents the interface of system i queried at $u \in U$. The components of each $q_i(u)$ are values determined by constraints that are not exposed at the interface. The final element of the interface is an element of $\{0,1\}$ that indicates whether a system is active (1) or not (0) by evaluating a constraint. To describe composable systems, we are provided attachment maps that indicate how the systems may interact. Systems are capable of interaction if they are composable, i.e. if component values match at the interfaces after attachment. For example, consider queries q_1, q_4 with attachment maps

$$a_1 = a_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The composition $a_1 \circ q_1 = a_4 \circ q_4$ because

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} Q \\ A \\ B \\ 1 \end{bmatrix} = \begin{bmatrix} Q+B \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} B \\ G \\ Q \\ 1 \end{bmatrix} \tag{7}$$

Equation 7 implies $\mathsf{D}_1 \rightarrow \mathsf{D}_{14} \leftarrow \mathsf{D}_4$ is well defined, therefore design spaces $\mathsf{D}_1 (\cong \mathbb{R}^4)$ and $\mathsf{D}_4 (\cong \mathbb{R}^4)$ represented as vertices may be joined by an edge in a simplicial complex. Similarly we can define other interactions between design spaces of active systems through attachment maps. All feasible interactions in this example are defined in terms of the cell complex in Figure 3.

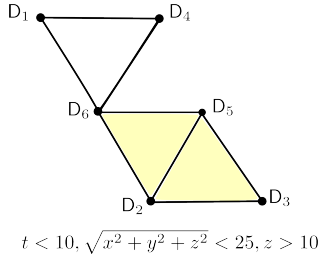


Fig. 3. Feasible interactions between active systems. A $k-$dimensional simplex implies $k-$composable interfaces.

Notice that if $t > 10$ in $q_1(u)$, S^1 and S^4 are not composable because S^1 is inactive but S^4 is active (and the sheaf condition is lost). Similarly S^1 and S^6 are not composable when S^1 is inactive. Assuming all other systems other than S^1 remain active, the simplicial complex may be updated to reflect this change through an elementary collapse that removes D_1. This results in the updated cell complex shown in Figure 4.

Observe that n constraints on the system of systems lead to $n!$ distinct simplicial complexes with associated cellular

[1]The set of all higher dimensional simplices incident to a vertex is called its *star*, so collapsing a vertex removes its star from the simplicial complex

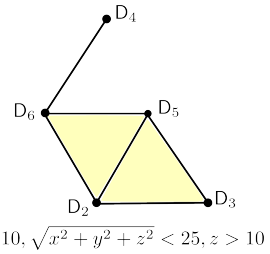


Fig. 4. The simplicial complex in Figure 3 after collapsing D_1. Notice from Figure 3 that the star at the vertex D_1 consisted solely of edges. If a vertex such as D_3 is collapsed, it would remove two edges and a triangle from the simplicial complex.

sheaves. Furthermore, observe that any of these sheaves may be transformed to any of the other $(n!-1)$ remaining sheaves through a sequence of elementary collapses and gluing operations. Then notice that the gluing and collapsing operations are inverse maps of each other, in the sense that a simplicial complex remains unchanged if a vertex is collapsed and then glued back or vice versa. Therefore all compositions of active systems through design spaces may be systematically generated and tested. The cellular sheaf corresponding to each topology encodes the interface compositions and the design space for the composed system of systems. The problem of system composition is transformed into the dual problem of interface composition through attachment maps, which leads to the enumeration of all possible system of system topologies from constraints on queries.

When systems are inactive (or lost through a collapsing operation), we may wish to proxy their behavior using the remaining systems. In cases where the system of systems is engineered with redundancy (i.e. when an interface can be queried through many compositions), then the lost interface may be recovered by querying the data attached to the star of the vertex before collapse. This will return all active (uncollapsed) systems that share an interface with the inactive (collapsed) system. If the interface cannot be recovered by directly quering the cellular sheaf of the modified simplicial complex, a simulation of the remaining systems in all feasible compositions is required to identify the best composition that can approximate the lost system's behavior, while satisfying the stated constraints. Therefore the systematic enumeration of cellular sheaves is also an enumeration of all possible simulations that may be conducted to replicate collapsed system behavior.

V. Conclusions

We have shown that describing system interfaces in terms of a commutative diagram of queries leads to a data structure whose underlying topology is a simplicial complex, and where the data itself has the structure of a cellular sheaf. The proposed data structure is accompanied with elementary operations of gluing and collapse, which are essential to describe dynamically changing topology of interactions between multiple systems. The data structure also describes system models (through queries) and therefore captures the entire search space of possible compositions.

Further extensions to the proposed data structure would involve the application of tools from computational topology to predict system properties through topological invariants such as homology groups. A particularly interesting extension is the ability to combinatorially describe physical laws that govern the interaction of multiple systems, by extending the formulation of sheaves over simplicial complex to sheaves over a *chain complex*. We expect that using tools such as Tonti diagrams used in algebraic topological descriptions of physical theories will provide greater insight into finding stable interface compositions.

VI. Acknowledgment

The authors would like to thank John Paschkewitz from the Defense Sciences Office at DARPA for helpful discussions. Vadim Shapiro's research was supported by NSF grants CMMI-1344205 and CMMI-1361862 and the National Institute of Standards and Technology. The responsibility for errors and omissions lies solely with the authors.

References

[1] Drechsler, R., and Kühne, U., 2015. "Formal modeling and verification of cyber-physical systems".

[2] Rogers, G., and Bottaci, L., 1997. "Modular production systems: a new manufacturing paradigm". *Journal of Intelligent Manufacturing,* ***8***(2), pp. 147–156.

[3] Sztipanovits, J., Koutsoukos, X., Karsai, G., Kottenstette, N., Antsaklis, P., Gupta, V., Goodwine, B., Baras, J., and Wang, S., 2012. "Toward a science of cyber–physical system integration". *Proceedings of the IEEE,* ***100***(1), pp. 29–44.

[4] Gössler, G., and Sifakis, J., 2003. *Formal Methods for Components and Objects: First International Symposium, FMCO 2002, Leiden, The Netherlands, November 5-8, 2002, Revised Lectures*. Springer Berlin Heidelberg, Berlin, Heidelberg, ch. Composition for Component-Based Modeling, pp. 443–466.

[5] Geisberger, E., and Broy, M., 2015. *Living in a networked world: Integrated research agenda Cyber-Physical Systems (agendaCPS)*. Herbert Utz Verlag.

[6] Robinson, M., 2014. *Topological signal processing*. Springer.

[7] Joslyn, C., Hogan, E., and Capraro, M. C., 2015. "Conglomeration of heterogeneous content using local topology (chclt)".

[8] Edelsbrunner, H., and Harer, J., 2010. *Computational topology: an introduction*. American Mathematical Soc.

[9] Desbrun, M., Kanso, E., and Tong, Y., 2008. "Discrete differential forms for computational modeling". In *Discrete differential geometry*. Springer, pp. 287–324.

[10] Ramaswamy, V., and Shapiro, V., 2003. "Combinatorial laws for physically meaningful design". In ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. 585–594.

[11] Tonti, E., 2001. "A direct discrete formulation of field laws: The cell method". *CMES- Computer Modeling in Engineering and Sciences,* ***2***(2), pp. 237–258.

[12] Goguen, J. A., 1992. "Sheaf semantics for concurrent interacting objects". *Mathematical Structures in Computer Science,* ***2***(02), pp. 159–191.

[13] Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F., 2007. "Service-oriented computing: State of the art and research challenges". *Computer*(11), pp. 38–45.

[14] De Giacomo, G., Di Ciccio, C., Felli, P., Hu, Y., and Mecella, M., 2012. "Goal-based composition of stateful services for smart homes". In *On the Move to Meaningful Internet Systems: OTM 2012*. Springer, pp. 194–211.

[15] De Giacomo, G., Mecella, M., and Patrizi, F., 2014. "Automated service composition based on behaviors: The roman model". In *Web Services Foundations*. Springer, pp. 189–214.

[16] Muscholl, A., and Walukiewicz, I., 2007. "A lower bound on web services composition". In *Foundations of Software Science and Computational Structures*. Springer, pp. 274–286.