## (1) Overview

### Title
**The Hetero-functional Graph Theory Toolbox**

### Paper Authors
1. Thompson, Dakota J.[1,2,*]
2. Hegde, Prabhat[1]
3. Schoonenberg, Wester C. H.[1]
4. Khayal, Inas S.[2]
5. Farid, Amro M.[1]

### Paper Author Roles and Affiliations
1. Thayer School of Engineering at Dartmouth, Hanover, NH, 03755, USA
2. The Dartmouth Institute Geisel School of Medicine at Dartmouth, Lebanon, NH, 03766, USA

*Dakota.J.Thompson.TH@Dartmouth.edu

### Abstract
In the 20th century, the analysis, design, planning, and operation of large-scale heterogeneous engineering systems from a holistic perspective has necessitated evermore sophisticated modeling techniques. Hetero-Functional Graph Theory (HFGT) has emerged as a means to address the complexity of engineering systems. This recently developed *Hetero-functional Graph Theory Toolbox* facilitates the computation of HFGT mathematical models and provides a Graphical User Interface based Petri net simulator to visualize the mathematical models. It is written in the MATLAB language and has been tested with v9.6 (R2019a). It is openly available on GitHub with a sample input file for straightforward re-use.

### Keywords
Hetero-functional Graph Theory; Model Based Systems Engineering; Petri nets

### Introduction
In the 20th century, newly invented technical artifacts products were connected to form large-scale complex engineering systems. Indeed, the electric generator, the telephone, the petro-chemical refinery, the automobile and a whole host of medical treatment and imaging devices have given rise to the electric power, communication, oil & gas pipeline, transportation, and healthcare delivery systems that we know today [1]. Over time, these engineering systems have evolved to incorporate newer technologies that rely on multiple engineering systems (e.g. renewable energy, mobile phones, fuel-cells, electric-vehicles, and wearable-health technologies).

Consequently, the interactions found within these networked systems have grown in both degree as well as heterogeneity. Furthermore, these already complex engineering systems have *converged* in what is now called *systems-of-systems*. The "smart" grid, the energy-water nexus, electrified transportation systems, the energy-water-food nexus, and the development of interdependent smart-city infrastructure are all examples of how contemporary systems-of-systems are becoming an integral part of human life [2]. This paper regards such systems as engineering systems:

**Definition 1.** Engineering System [1] : A class of systems characterized by a high degree of technical complexity, social intricacy, and elaborate processes aimed at fulfilling important functions in society.

The analysis, design, planning, and operation of these engineering systems from a holistic perspective has necessitated ever-more sophisticated modeling techniques. Two informatic sciences are of particular relevance. ***Model-Based Systems Engineering*** (MBSE) is a practical and interdisciplinary engineering field that enables a successful realization of complex systems from concept, through design, to full implementation[3]. Often, the practice of MBSE relies on the use of UML[4] and/or SysML[5, 6] which consists of several *graphical* viewpoints of the function and form of the engineering system[7]. These viewpoints include block diagrams, activity diagrams, and state-machine diagrams; making them well-equipped to describe the complex and heterogeneous nature of these systems; both graphically and later in simulation. In the meantime, ***network science*** has matured as a discipline to provide quantitative analyses for the structure and function of networks that appear across natural, social and engineering sciences[8, 9]. The spatially-distributed nature of engineering systems has often times led to their underlying models being rooted in graph theory. Thus, it has been applied to systems like transportation systems[10], power grids[11, 12], water networks[13], supply chains[14], and healthcare systems[15].

Despite significant advancements, these seemingly disparate fields have experienced similar limitations in addressing the inherent complexity of engineering systems. While the graphical models used in MBSE often serve as the basis for developing complex simulations of *system behavior*, they fall short in providing a quantitative analysis of *system structure*. On the other hand, network science's reliance on graphs as a data structure limits its ability to handle the explicit heterogeneity one often encounters in engineering systems. Even the recent developments toward multi-layer networks have been shown to exhibit several modeling constraints that inhibit the representation of an arbitrary number of network layers of arbitrary topology connected arbitrarily[16, 17]. The trend toward greater and more heterogeneous interaction between multiple engineering systems is only set to accelerate given the continual proliferation of diverse physical and information technologies[18, 19, 20, 21, 22]. The current methodological and theoretical limitations in the MBSE and network science fields necessitates new mathematical modeling techniques for multiple integrated engineering systems.

Hetero-Functional Graph Theory (HFGT) has emerged as a means to model the structure and function of highly interconnected and heterogeneous engineering systems [17]. In order to support insightful quantitative analyses, HFGT relies on multiple graphs as data structures. It also explicitly incorporates the heterogeneity of conceptual and ontological constructs found in MBSE. In doing so, it facilitates the translation of systems engineering models (e.g. SysML) to a mathematical engineering systems description, providing a rigorous platform for modeling systems-of-systems. The foundational HFGT works are in the field of mass-customized production systems [23, 24, 25, 26]. In some ways, such production systems present modelling challenges that are common to most engineering systems. The production capabilities of a production system can come together in various permutations and combinations to produce an almost infinite number of product variants. At the same time, their structure and behavior is dynamically changing. Since these first works, the theory has methodologically evolved to provide qualitative and quantitative analyses in other application domains including electric power systems[27, 28], energy-water nexus [29, 30, 31, 17], electrified transportation systems[32, 33, 34], microgrid-enabled production systems[35], industrial energy management[36], personalized healthcare delivery systems[37, 38, 39] and interdependent smart city infrastructures[17].
Hetero-functional graph theory is composed of seven mathematical models that together form a System Adjacency Matrix $\mathbb{A}$ as its eighth.

1. System Concept $A_S$
2. Hetero-functional Adjacency Matrix $A_\rho$
3. Hetero-functional Incidence Tensor $\mathcal{M}_\rho$
4. Controller Agency Matrix $A_Q$
5. Controller Adjacency Matrix $A_C$
6. Service as Operand Behavior $N_L$
7. Service Feasibility Matrix $\Lambda$

This paper serves as a user guide to a recently developed *Hetero-functional Graph Theory Toolbox*[40] which facilitates the instantiation of these seven mathematical models from a single XML input file. It is written in the MATLAB[41] language and has been tested with v9.6 (R2019a). It is openly available on GitHub[40] together with a sample input XML file for straightforward re-use. Additionally this paper serves as a user guide to a Graphical User Interface (GUI) that facilitates the use of the HFGT toolbox and an event-driven Comma Separated Value (CSV) file to visualize flows of operands through the HFG.

The remainder of the paper proceeds as follows. The section entitled: "Creating a HFGT Input File" details how to translate a real-world system into an HFGT Toolbox input XML file. The following section entitled "HFGT Data Structures and Functions" describes the myLFES (LFES: Large Flexible Engineering System) data structure which organizes all of the data generated by the toolbox. This section also describes the functions that transform the input XML file into an empty version of the myLFES data structure. Lastly, it describes the functions that populate the myLFES data structure with HFGT values. The section entitled "Toolbox validation" describes how the HFGT toolbox has been validated against previously published results.The next section entitled "Petri net Visualization" describes the additional input information required for visualizing the HFG in a Petri net. This section also describes how the myPetriNetwork system structure is structured. Finally, the "Conclusions" section brings the paper to a close. This paper presumes that the reader has a working knowledge of HFGT which is otherwise gained from a thorough reading of the associated text [17]. It is also assumed that the reader has a working knowledge of Petri nets which is otherwise gained from the associated text [42, 43, 44]. Furthermore, the HFGT toolbox uses an XML input file and so a basic knowledge of the eXtensible Markup Language[45] is needed. Additionally, the Tensor Toolbox for MATLAB [46, 47] is used to store incidence tensors. Finally, UML/SysML models[5] are used to convey the object-oriented programming data structures found throughout the HFGT toolbox.

## Implementation and architecture

### Creating a HFGT XML Input File

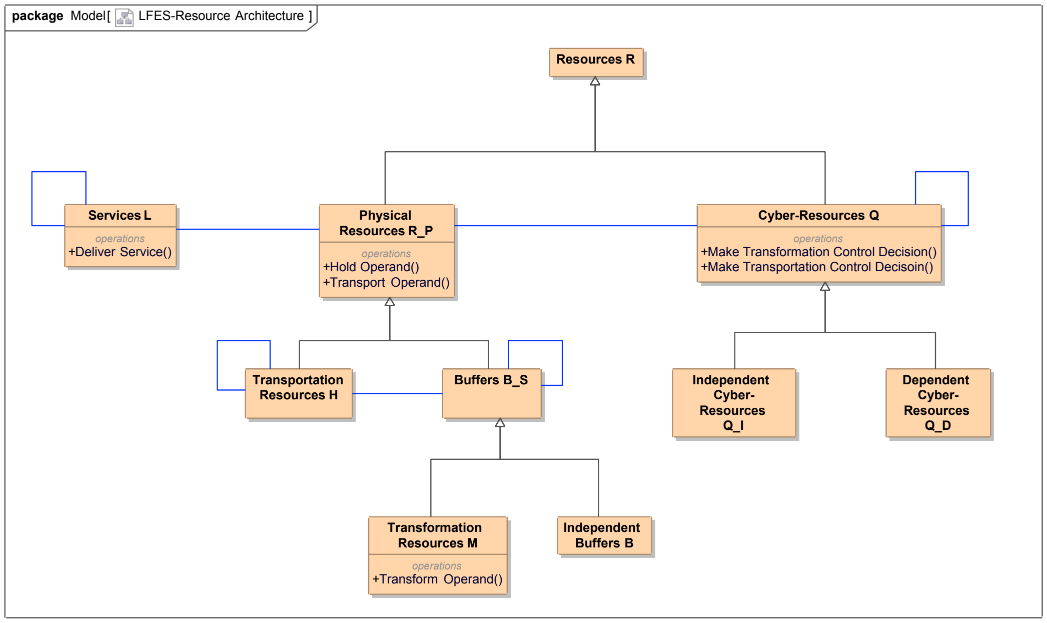
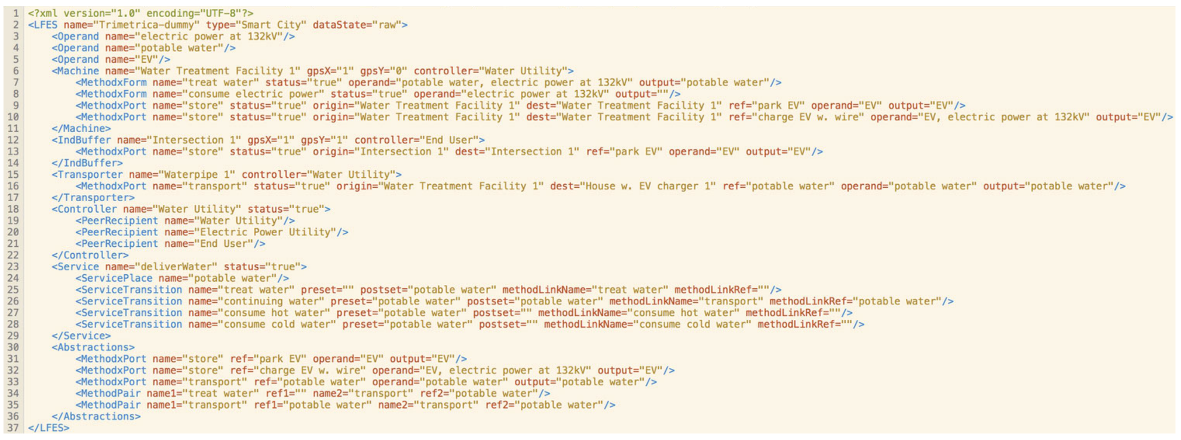

package Model[ LFES-Resource Architecture ]
Resources R
Services L
operations
+Deliver Service()
Physical Resources R_P
operations
+Hold Operand()
+Transport Operand()
Cyber-Resources Q
operations
+Make Transformation Control Decision()
+Make Transportation Control Decisoin()
Transportation Resources H
Buffers B_S
Independent Cyber-Resources Q_I
Dependent Cyber-Resources Q_D
Transformation Resources M
operations
+Transform Operand()
Independent Buffers B

Figure 1: A SysML Block Diagram of the System Form of the LFES Meta-Architecture



```xml
<?xml version="1.0" encoding="UTF-8"?>
<LFES name="Trimetrica-dummy" type="Smart City" dataState="raw">
    <Operand name="electric power at 132kV"/>
    <Operand name="potable water"/>
    <Operand name="EV"/>
    <Machine name="Water Treatment Facility 1" gpsX="1" gpsY="0" controller="Water Utility">
        <MethodxForm name="treat water" status="true" operand="potable water, electric power at 132kV" output="potable water"/>
        <MethodxForm name="consume electric power" status="true" operand="electric power at 132kV" output=""/>
        <MethodxPort name="store" status="true" origin="Water Treatment Facility 1" dest="Water Treatment Facility 1" ref="park EV" operand="EV" output="EV"/>
        <MethodxPort name="store" status="true" origin="Water Treatment Facility 1" dest="Water Treatment Facility 1" ref="charge EV w. wire" operand="EV, electric power at 132kV" output="EV"/>
    </Machine>
    <IndBuffer name="Intersection 1" gpsX="1" gpsY="1" controller="End User">
        <MethodxPort name="store" status="true" origin="Intersection 1" dest="Intersection 1" ref="park EV" operand="EV" output="EV"/>
    </IndBuffer>
    <Transporter name="Waterpipe 1" controller="Water Utility">
        <MethodxPort name="transport" status="true" origin="Water Treatment Facility 1" dest="House w. EV charger 1" ref="potable water" operand="potable water" output="potable water"/>
    </Transporter>
    <Controller name="Water Utility" status="true">
        <PeerRecipient name="Water Utility"/>
        <PeerRecipient name="Electric Power Utility"/>
        <PeerRecipient name="End User"/>
    </Controller>
    <Service name="deliverWater" status="true">
        <ServicePlace name="potable water"/>
        <ServiceTransition name="treat water" preset="" postset="potable water" methodLinkName="treat water" methodLinkRef=""/>
        <ServiceTransition name="continuing water" preset="potable water" postset="potable water" methodLinkName="transport" methodLinkRef="potable water"/>
        <ServiceTransition name="consume hot water" preset="potable water" postset="" methodLinkName="consume hot water" methodLinkRef=""/>
        <ServiceTransition name="consume cold water" preset="potable water" postset="" methodLinkName="consume cold water" methodLinkRef=""/>
    </Service>
    <Abstractions>
        <MethodxPort name="store" ref="park EV" operand="EV" output="EV"/>
        <MethodxPort name="store" ref="charge EV w. wire" operand="EV, electric power at 132kV" output="EV"/>
        <MethodxPort name="transport" ref="potable water" operand="potable water" output="potable water"/>
        <MethodPair name1="treat water" ref1="" name2="transport" ref2="potable water"/>
        <MethodPair name1="transport" ref1="potable water" name2="transport" ref2="potable water"/>
    </Abstractions>
</LFES>
```

Figure 2: An example HFGT Toolbox input XML input file

As a high-level overview, the purpose of the HFGT input file is to provide a structured representation of the data associated with an instantiated large flexible engineering systems. To that end, Figure 1 shows the meta-architecture of the system form of a Large Flexible Engineering System (LFES) as described by HFGT[17]. HFGT assumes that the formal elements of any LFES can be viewed as instances of this meta-architecture. The input data file, thus, organizes the resources present in an engineering system into these "meta-elements" and captures the functions (i.e. methods) that they are capable of performing. Such a complex hierarchical structure is readily stated in the eXtensible Markup Language (XML)[45]. Furthermore, XML files are both human and machine readable; making them easy to check for inadvertently introduced errors.

The first step to using the HFGT toolbox is to accurately write an input XML file that instantiates the meta-architecture shown in Figure 1. Figure 2 provides an example of such an XML file. Each line of the XML file is composed of a start or end XML element (e.g <LFES>) that corresponds to one of the classes identified in Figure 1. Furthermore, each of these elements has one or more attributes that correspond to the methods and attributes of these classes. Each of these XML elements are now explained in turn.

**LFES**: This element is the root of the input XML file and indicates an instantiated LFES. All of the meta-elements shown in Figure 1, and the processes that they are capable of carrying out are contained within this root. The attributes "name" and "type" describe the name of the system and its type. The attribute "dataState" is defaulted to "raw" to indicate its pre-processed state.

**Machine**: This XML element is representative of a "Transformation Resource" (M) shown in Figure 1. The "name" attribute describes the name of the transformation resource. If a controller has agency over the transformation resource, the former's name is stored under the "controller" attribute. The attributes "gpsX" and "gpsY" capture the physical location of the transformation resource with respect to a reference axes. The illustrative example in Figure 2 shows a water treatment facility as an instantiaton of a transformation resource.

**MethodxForm**: This XML element is used to describe the ability of a transformation resource to transform an operand. As shown in Figure 1 and as reflected in the illustrative example in Figure 2, this element is nested within the Machine (i.e. transformation resources) element. The attributes “name”, “status”, “operand”, and “output” describe the name of the process, its status (active/inactive), the set of operands needed for the process, and the set of outputs as a result of the transformation respectively.

**MethodxPort**: As shown in Figure 1, the operations involving holding or transportation of an operand are universal to all of the physical resources. Rather than introduce an XML element for holding and transportation processes separately, the XML file format introduces this element to refer to a “refined transportation process"[17] as their combination. Furthermore, a storage process is considered a transportation with the same origin and destination. The attributes used to describe these process include “name”, “status”, "origin", and "dest", which describe the name, the active state, the origin and the destination of the refined transportation process respectively. The attribute “ref” describes the refinement of the transportation in terms of a string that reflects the associated holding process. The operands needed to carry out the process and the output of the process are described by the attributes “operand” and “output” respectively.

**IndBuffer**: This XML element is representative of an “Independent Buffer” (B) as shown in Figure 1. The “name” attribute describes the name of the independent buffer. If a controller has agency over the independent buffer, the former’s name is stored under the “controller” attribute. The attributes “gpsX” and “gpsY” capture the physical location of the independent buffer with respect to a reference axes.

**Transporter**: This XML element represents a “Transportation Resource” (H) as shown in Figure 1. The attributes “name” and “controller” captures its name and the controller that controls the resource respectively.

**Controller**: As shown in Figure 1, controllers have jurisdiction or agency over physical resources. The association of a physical resource to an independent controller is captured in the attributes of the physical resource itself. Dependent controllers embedded within a physical resource are implicit to the tooolbox’s functionality and are not explicitly stated in the input XML File. Thus, the “controller” XML element describes independent controllers exclusively. The attribute “name” describes the name of the controller and the attribute “status” describes whether the controller is active or inactive.

**PeerRecipient**: This XML element describes the peer controllers that *receive information from* the controller named in the controller XML element. The name attribute provides the name of this peer controller.

**Service**: The services shown in Figure 1 describe the evolution in the state of an operand as a delivered service modeled as a Petri Net[48]. The attribute "name" describes the name of the service and the attribute "status" describes whether the service is active or inactive in the system.

**ServicePlace**: This XML element describes the places of the service (Petri) net. Each place is given its own name under the "name" attribute.

**ServiceTransition**: This XML element describes the transitions of the service (Petri) net. Each transition is given its own name under "name" attribute. The "preset" and "postset" attributes refer to the respective names of the places that send tokens to or receive tokens from the service transition named in this XML element. The "methodLinkName" attribute indicates the name of the system process to which the transition is linked. The "methodLinkRef" attribute indicates the refinement of the transportation process to which the transition is linked (if any). As shown in Figure 1, services and physical resources are associated by virtue of the link between the service transitions in the former and the system processes provided by the latter.

**Abstractions**: This XML element has no attributes associated with it. It contains all possible holding refinements and functional sequences (i.e. MethodPairs) in the system nested within it.

**MethodxPort** (within the Abstractions Element): This XML element contains information about all possible refinements and their associated operands and outputs in the system. In the illustrative example in Figure 2, the "refinements" possible while transporting the Electric Vehicle (EV) are: parking the EV, charging it by wire, discharging it and charging it wirelessly.

**MethodPair** (within the Abstractions Element): This XML element captures the possibility of one system process following another in the system. In Figure 2, the possibility that water being transported from an origin to a destination can be followed by a transportation from the new origin to a new destination has been described.

## HFGT Data Structures and Functions

The *Hetero-functional Graph Theory Toolbox* uses MATLAB's object-oriented programming functionality to enhance modularity, extensibility and reusability. The primary purpose of the HFGT toolbox is to instantiate and subsequently populate the data in the `myLFES` structure. It's associated class diagram is shown in Figure 3. The `myLFES` stores all of the information in the XML input file and then calculates all of the mathematical quantities identified in HFGT. This high-level purpose is achieved through two principal modules: `XML2LFES()` and `raw2FullLFES()` that are executed in sequence as shown in Figure 4. In brief, the `XML2LFES()` module serves to import the input XML file and create the `myLFES` data structure in a "raw" structure. Then, the `raw2Full()` module makes the HFGT calculations necessary to convert the `myLFES` data structure to the "full" state. Each of these modules is now discussed in detail.

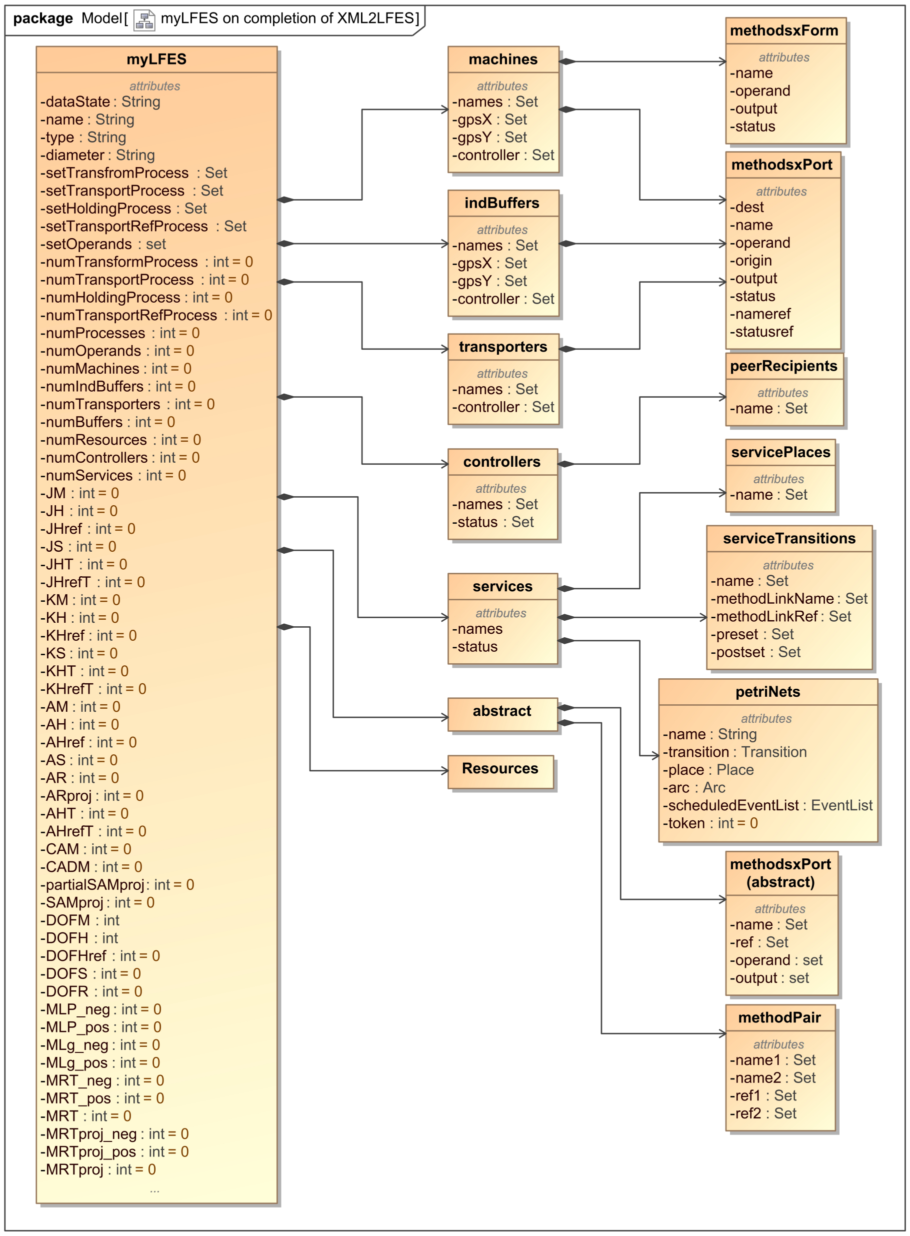


Figure 3: Structure of myLFES on completion of XML2LFES

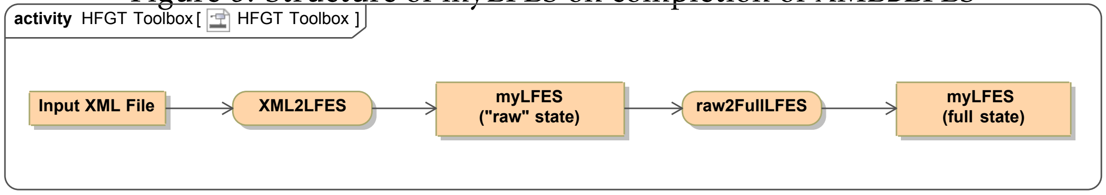


Figure 4: A high-level activity diagram of the HFGT Toolbox

## [myLFES,S] = XML2LFES(XMLFile)

As previously mentioned, the `XML2LFES()` module serves to import the input XML file and create the `myLFES` data structure in a “raw" structure. This functionality is achieved through the sequence of functions shown in the activity diagram in Figure 5. Each of these functions are now explained in turn.

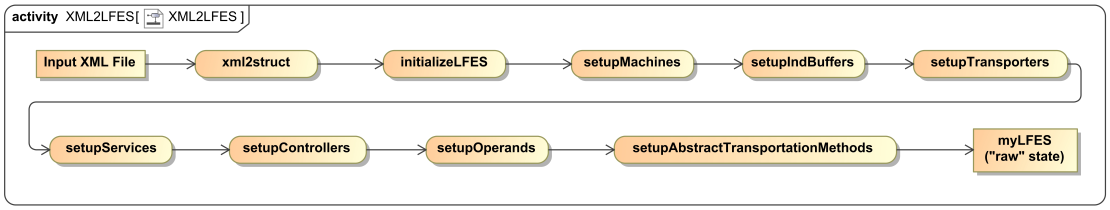


Figure 5: Activity diagram of the XML2LFES module

**S = xml2struct(xmlfile)** : The `xml2struct` function (v1.8.0.0) is a freely available third-party MATLAB function that can be otherwise downloaded from the MATLAB Central website[49]. In the context of the HFGT toolbox, this function reads the input XML file and converts it into an intermediate object `S` which replicates the hierarchy of the input XML file. The `S` structure is identical to the `myLFES` structure shown in Figure 3 with two exceptions:

1. It does not contain the root attributes of myLFES. These are calculated later.
2. At the lowest level of decomposition, `S` has a cell array of objects (e.g. `S.myLFES.machines{:}.m`
   In `myLFES`, the lowest level of decomposition has an object of cell arrays (e.g. `myLFES.machines...`
   `.methodsxForm{:}`).

Because the `xml2struct()` function is relatively rigid, and there is not much control on its output `S`, the remaining functions in `XML2LFES()` serve to "reshape" `S` into the more desirable form `myLFES` without any loss of information.

**myLFES = initializeLFES()** : This function is the LFES constructor and consequently instantiates `myLFES`. The high-level integer attributes of `myLFES` are initialized to zero and the sub-classes are initialized as empty structured arrays. It also calculates the number of transformation and structural degrees of freedom in `myLFES.DOFM` and `myLFES.DOFH` respectively.

**myLFES = setupMachines(myLFES, S)** : This function assigns information to the sub-object `myLFES.machines` (M) by "unpacking" `S.myLFES.machines`. It also adds the `nameref` and `statusref` attributes to the `myLFES.machines... .methodsxPort` ($P_\eta$) class. These refer to the name of the associated transportation process and its initial status respectively. This function also collates the set of all transformation processes ($P_\mu$) in `myLFES.setTransformProcess`.

**myLFES = setupIndBuffers(myLFES, S)** :This function assigns information to the sub-object `myLFES.indBuffers` (B) by "unpacking" `S.myLFES.indBuffers`. It also adds the `nameref` and `statusref` attributes to the `myLFES. ...indBuffers.methodsxPort`

($P_\eta$) class. These refer to the name of the associated transportation process and its initial status respectively. It also increments the value of myLFES.DOFH with the capabilities of the independent buffers.

**myLFES = setupTransporters(myLFES, S)** : This function assigns information to the sub-object myLFES...
.transporters (H) by "unpacking" S.myLFES.transporters. It also adds the nameref and statusref attributes to the myLFES... transporters.methodsxPort ($P_\eta$) class. These refer to the name of the associated transportation process and its initial status respectively. It also increments the value of myLFES.DOFH with the capabilities of the transportation resources.

**myLFES = setupServices(myLFES, S)** : This function assigns information to the sub-object myLFES.services by "unpacking" S.myLFES.services.

**myLFES = setupControllers(myLFES, S)** : This function assigns information to the sub-object myLFES.controllers (Q) by "unpacking" S.myLFES.controllers.

**myLFES = setupOperands(myLFES, S)** : This function assigns information to the set myLFES.setOperands (L) and myLFES.numOperands by "unpacking" S.myLFES.Operand.

**setupAbstractTransportationMethods(myLFES, S)** : This function assigns information to the sub-object myLFES.
..abstract by "unpacking" S.myLFES.abstract.

The following support functions are used within the functions that set up the elements of the system:

**objB=getResourceAttributes(objA)** : This function copies the attributes of a resource from the intermediate structure S to the corresponding resource class object within the myLFES object. This is helpful in specifying the attributes of each type of meta-element within myLFES.

**[objB, setProcess, DOF]=getResourceMethods(objA,setProcess,DOF,opt)** : This function copies information associated with a resource from the intermediate structure S to the corresponding resource class object within the myLFES object. The opt argument of the function follows the classification in Figure 2. It is assigned "M","B" or "H" for machines, independent buffers and transporters respectively. In addition, it computes the transportation and transportation capabilities.

**objA=insertObjB(objA,objB,idxA)** : This function inserts the fields of objB into the equivalent fields of objA. This function is used as a support function within both getResourceAttributes and getResourceMethods to extract fields from structure S and fit it into the correct hierarchical location within myLFES.

## myLFES=raw2FullLFES(myLFES)

As mentioned previously, the raw2FullLFES() module makes the HFGT calculations necessary to convert the myLFES data structure from the "raw" to the "full" state. This functionality is achieved through the sequence of functions shown in the activity diagram in Figure 6. Each of these functions are now explained in turn.

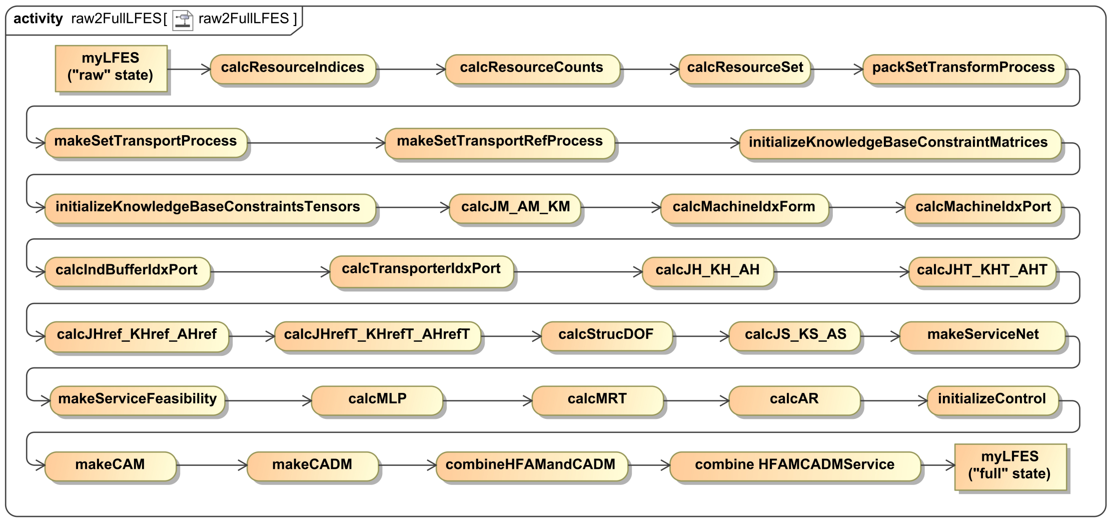


Figure 6: Activity diagram of the raw2Full module

**myLFES=calcResourceIndices(myLFES)** : This function assigns the new attributes `idxMachine`, `idxBuffer` and `idxTransporter` to the sub-objects `myLFES.machines` (M), `myLFES.indBuffers` (B), and `myLFES.transporters` (H) respectively. These attributes represent the numeric index of the resource for each type of physical resource in the system. In addition, it also assigns an attribute `idxResource` to the sub-objects `myLFES.machines` (M), `myLFES.indBuffers` (B), and `myLFES.transporters` (H) which indicates the index of the resource in the set of resources.

**myLFES=calcResourceCounts(myLFES)** : This function calculates the number of each type of resource present in the system. In doing so, it populates the following attributes: `myLFES.numMachines`, `myLFES.numIndBuffers`, `myLFES.numBuffers`, `myLFES.numTransporters`, `myLFES.numResources`, `myLFES.numControllers` and `myLFES.`
`...numServices`.

**myLFES=calcResourceSet(myLFES)** : This function creates a new class object `myLFES.resources` with two attributes: `names` and `idx`. These are the set unions of the `names` and `idxResource` attributes in `myLFES.machines` (M), `myLFES.indBuffers` (B) and `myLFES.transporters` (H).

**myLFES=packSetTransformProcess(myLFES)** : This function computes the set of unique transformation processes and assigns it to `myLFES.setTransformProcess` ($P_\mu$). It also assigns the number of unique transformation processes to `myLFES.numTransformProcess`

**myLFES=makeSetTransportProcess(myLFES)** : This function computes the set of all possible transportation processes system and assigns it to `myLFES.setTransportProcess` ($P_\eta$). It also assigns the number of transportation processes to `myLFES.numTransportProcess`.

**myLFES=makeSetTransportRefProcess(myLFES)** : This function computes the set of all possible refined transportation processes and assigns it to `myLFES.setRefTransportProcess` ($P_{\bar{\eta}}$). It also uses the information in `myLFES.abstract`$\{p_{\gamma g}\}$ `...methodsxport` to compute the set of holding processes and assign the result to `myLFES.setHoldingProcess` ($P_\gamma$). It also computes the numbers of refined transportation processes and holding processes and assigns them to `myLFES.numTransportProcess` and `myLFES.numHoldingProcess` respectively.

**myLFES=initializeKnowledgeBasesConstraintsMatrices(myLFES)** : This function creates appropriately sized but empty sparse matrices to the three knowledge bases (i.e. `myLFES.JM` ($J_M$), `myLFES.JH` ($J_H$), `myLFES.JHRef`) ($J_{\bar{H}}$), the three constraint matrices (i.e. `myLFES.KM` ($K_M$), `myLFES.KH` ($K_H$), `myLFES.KHRef`) ($K_{\bar{H}}$), and the three system concept matrices (i.e. `myLFES.AM` ($A_M$), `myLFES.AH` ($A_H$), `myLFES.AHRef` ($A_{\bar{H}}$)).

**myLFES=initializeKnowledgeBasesConstraintsTensors(myLFES)** : This function creates appropriately sized but empty sparse tensors to two knowledge bases (i.e. `myLFES.JHT` ($\mathcal{J}_H$), `myLFES.JHRefT` ($\mathcal{J}_{\bar{H}}$)), the two constraint tensors (i.e. `myLFES.KHT` ($\mathcal{K}_H$), `myLFES.KHRefT` ($\mathcal{K}_{\bar{H}}$)), and the two system concept tensors (i.e. `myLFES.AHT` ($\mathcal{A}_H$), `myLFES.AHRefT` ($\mathcal{A}_{\bar{H}}$)).

**myLFES=calcJM_KM_AM(myLFES)** : This function computes the transformation knowledge base (`myLFES.JM`, $J_M$), the transformation constraint matrix (`myLFES.KM`, $K_M$) and the resultant transformation system concept (`myLFES.AM`, $A_M$).

**myLFES=calcMachineIdxPort(myLFES)** : This function computes the index associated with the transformation process conducted by a given machine. It thus populates `myLFES.machines.methodsxForm.idxForm`, the index of the transformation process from the set of transformation processes possible in the system.

**myLFES=calcMachineIdxPort(myLFES)** : This function computes the following five indices associated with a refined transportation process conducted by a given machine.

- `myLFES.machines.methodsxPort.idxOrigin` : the index of the resource where the transportation/holding process originates from the list of indices of all resources present in the system.
- `myLFES.machines.methodsxPort.idxDest` : the index of the resource where the transportation/holding process terminates from the list of indices of all resources present in the system.
- `myLFES.machines.methodsxPort.idxHold` : the index of the holding process from the list of holding processes in the system.

- `myLFES.machines.methodsxPort.idxPort` : the index of the transportation process from the set of transportation processes possible in the system.
- `myLFES.machines.methodsxPort.idxPortRef` : the index of the refined transportation process from the set of refined transportation processes possible in the system.

**myLFES=calcIndBufferIdxPort(myLFES)**: This function is analogous to `calcMachineIdxPort()` above, but instead calculates the index attributes in `myLFES.indBuffers.methodsx`

**myLFES=calcTransporterIdxPort(myLFES)**: This function is analogous to `calcMachineIdxPort()` above, but instead calculates the index attributes in `myLFES.transporters.method`

**myLFES=calcJH_KH_AH(myLFES)** : This function computes the transportation knowledge base matrix (`myLFES.JH`, $J_H$), the transportation constraint matrix (`myLFES.KH`, $K_H$) and the transportation system concept matrix (`myLFES.AH`, $A_H$).

**myLFES=calcJHT_KHT_AHT(myLFES)** : This function computes the transportation knowledge base tensor (`myLFES.JHT`, $\mathcal{J}_H$), the transportation constraint tensor (`myLFES.KHT`, $\mathcal{K}_H$) and the transportation system concept tensor (`myLFES.AHT`, $\mathcal{A}_H$).

**myLFES=calcJHref_KHref_AHref(myLFES)** : This function computes the refined transportation knowledge base
(`myLFES.JHref`, $J_{\bar{H}}$), the refined transportation constraint matrix (`myLFES.KHref`, $K_{\bar{H}}$) and the refined transformation system concept (`myLFES.AHref`, $A_{\bar{H}}$).

**myLFES=calcJHrefT_KHrefT_AHrefT(myLFES)** : This function computes the refined transportation knowledge base tensor (`myLFES.JHrefT`, $\mathcal{J}_{\bar{H}}$), the refined transportation constraint tensor (`myLFES.KHrefT`, $\mathcal{K}_{\bar{H}}$) and the refined transportation system concept tensor (`myLFES.AHrefT`, $\mathcal{A}_{\bar{H}}$).

**myLFES=calcStrucDOF(myLFES)** : This function calculates the number of independent actions that completely define the number of available transformation, transportation and refined transportation processes in a system and assigns it to `myLFES.DOFM`, `myLFES.DOFH`, `myLFES.DOFHref` respectively.

**myLFES=calcJS_KS_AS(myLFES)** : This function computes the system knowledge base (`myLFES.JS`, $J_S$), the system constraint matrix (`myLFES.KS`, $K_S$) and the system concept (`myLFES.AS`, $A_S$).

**myLFES=makeServiceNets(myLFES)** : This function captures information regarding the different states involved in delivering an operand in the system and translates it into a service Petri net ($N_{l_i}$) with places ($S_{l_i}$), transitions ($\mathcal{E}_{l_i}$), and arcs ($M_{l_i}$) that defines the adjacency of the service activities in the system. For each service, it stores the positive incidence matrix, the negative incidence matrix and the dual-adjacency matrix under `myLFES.services.Mpos` ($M^+_{l_i}$), `myLFES.services.Mneg` ($M^-_{l_i}$) and `myLFES.services.dualAdjacency` ($A_{l_i}$) respectively.

**myLFES=makeServiceFeasibility(myLFES)** : This function computes the service feasibility matrix. It uses the structural degrees of freedom previously computed in the toolbox to compute the service degrees of freedom. It computes and assigns the following attributes to the `myLFES` object:

- `myLFES.services.rawLambda` ($\Lambda_i$): Service feasibility matrix in its original shape for each deliverable service.
- `myLFES.services.rawLambda_neg` ($\Lambda_i^-$): Negative component of the Service feasibility matrix in its original shape for each deliverable service.
- `myLFES.services.rawLambda_pos` ($\Lambda_i^+$): Positive component of the Service feasibility matrix in its original shape for each deliverable service.
- `myLFES.services.Lambda` ($\Lambda_{S_i}$): Service feasibility matrix projected to DOFs for each deliverable service.
- `myLFES.services.RawxFormLambda` ($\Lambda_{\mu i}$): Service transformation feasibility matrix in its original shape for each deliverable service.
- `myLFES.services.RawxFormLambda_neg` ($\Lambda_{\mu i}^-$): Negative Service transformation feasibility matrix in its original shape for each deliverable service.
- `myLFES.services.RawxFormLambda_pos` ($\Lambda_{\mu i}^+$): Positive Service transformation feasibility matrix in its original shape for each deliverable service.
- `myLFES.services.xFormLambda` ($\Lambda_{Mi}$): Service transformation feasibility matrix projected to system capabilities for each deliverable service.
- `myLFES.services.xPortLambda` ($\Lambda_{\gamma i}$): Service transportation feasibility matrix for each deliverable service.
- `myLFES.services.xPortLambda_neg` ($\Lambda_{\gamma i}^-$): Negative Service transportation feasibility matrix for each deliverable service.
- `myLFES.services.xPortLambda_pos` ($\Lambda_{\gamma i}^+$): Positive Service transportation feasibility matrix for each deliverable service.

**myLFES=calcMLP(myLFES)** : This function computes the positive and negative Holding Process-Operand Incidence Matrices `myLFES.MLg_neg` ($M_{LP_\gamma}^-$), `myLFES.MLg_pos` ($M_{LP_\gamma}^+$) and the positive and negative Process-Operand Incidence Matrices `myLFES.MLP_neg` ($M_{LP}^-$), `myLFES.MLP_pos` ($M_{LP}^+$).

**myLFES=calcMRT(myLFES)** : This function computes the positive and negative Hetero-functional Incidence Tensors `myLFES.MRT_neg` ($\mathcal{M}_\rho^-$), `myLFES.MRT_pos` ($\mathcal{M}_\rho^+$), `myLFES.MRT` ($\mathcal{M}_\rho$) and the positives and negative Projected Hetero-functional Incidence Tensors `myLFES.MRTproj_neg` ($\tilde{\mathcal{M}}_\rho^-$), `myLFES.MRTproj_pos` ($\tilde{\mathcal{M}}_\rho^+$), `myLFES.MRTproj` ($\tilde{\mathcal{M}}_\rho$).

**myLFES=calcAR(myLFES)** : This function computes the Hetero-Functional Adjacency Matrix `myLFES.AR` ($A_\rho$) and the projected Hetero-Functional Adjacency Matrix `myLFES.ARproj` ($\tilde{A}_\rho$). In parallel, the function calculates the number of physical continuity degrees of freedom and assigns their values to `myLFES.DOFR1`, `myLFES.DOFR2`,`myLFES. ...DOFR3`, and `myLFES.DOFR4` for Types 1 through 4 respectively. Furthermore, it computes the number of functional sequence dependent degrees of freedom and assigns it to `myLFES.DOFR5`. Finally, the function computes the total degree of freedom sequences in the system and stores it as `myLFES.DOFR`.

**myLFES = initializeControl(myLFES)** : This function creates appropriately sized but empty sparse matrices to the controller agency matrix (`myLFES.CAM`, $A_Q$), the controller adjacency matrix (`myLFES.CADM`, $A_C$) and the projected system adjacency matrix without services (`myLFES.partialSAMproj` $\tilde{\mathbb{A}}$)

**myLFES=makeCAM(myLFES)** : This function computes the controller agency matrix and assigns it to `myLFES.CAM` ($A_Q$).

**myLFES=makeCADM(myLFES)** : This function computes the controller adjacency matrix and assigns it to
`myLFES.CADM` ($A_C$).

**myLFES=combineHFAMandCADM(myLFES)** : This function computes the system adjacency matrix (without services) by combining the hetero-functional adjacency, the controller agency and the controller adjacency matrices. It is assigned to `myLFES.partialSAMproj` ($\tilde{\mathbb{A}}$).

**myLFES=combineHFAMCADMService(myLFES)** : This function combines the hetero-functional adjacency, controller agency, controller adjacency, and service graph matrices to produce the system adjacency matrix. This combined matrix is assigned to `myLFES.SAMproj` ($\tilde{\mathbb{A}}$).

## Toolbox Validation

The accurate production of a hetero-functional graph (HFG) as an instantiated mathematical model is similar to that of the creation of a formal graph based exclusively upon nodes and edges. Formal graph toolboxes are able to create formal graphs as instantiated mathematical models because they reproduce the results that would be achieved for small systems computed by hand. In other words, these toolboxes serve to automate the construction of these mathematical models that would be impractical to do by hand. Consequently, the HFGT toolbox has been validated by making sure that its results match the results found in a number of early HFGT publications[23, 24, 37, 27, 32, 33, 34, 50] where either manual or semi-automated methods were used. A wide variety of published test cases were tested so as to ensure that all of the HFGT toolbox data structures were instantiated and all of the HFGT toolbox methods were called.

## Petri Net Visualization

The HFGT toolbox is then used in the Petri net visualization GUI. This GUI takes the system modeled by the HFGT toolbox along with a scheduled event list and visualizes the flow of operands through the system via a Petri net. To visualize the system in a Petri net additional GPS locations and initial token counts need to be added to the input XML. To visualize the flow of operands through the system a scheduled event list needs to be created to describe the firing of DOFS and path of tokens.

## Creating a HFGT Petri net XML Input File

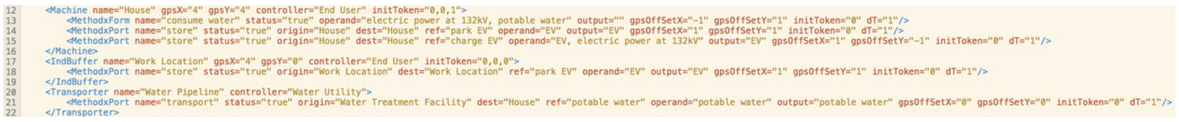
The XML required by the Petri net GUI for visualizing the HFG takes the same form and holds the same attributes as the XML used solely to create a HFG. However, it requires the addition of several attributes to `machines`, `independent buffers` and `processes`. `Machines` and `independent buffers` require the addition of initial token counts while all `processes` require the addition of initial token counts, GPS offset, and the process duration time. If the process is instantaneous of continuous in nature then the duration is set to 0. These are described here in turn.

**Machine**: This XML element is representative of a "Transformation Resource" (M) shown in Figure 1. The attribute "initTokens" needs to be added to describe the number of tokens present at the place when the system is initialized. The illustrative example in Figure 7 shows the addition of "initTokens" to a transformation resource.

**IndBuffer**: This XML element is representative of an "Independent Buffer" (B) as shown in Figure 1. The attribute "initTokens" needs to be added to describe the number of tokens present at the place when the system is initialized. The illustrative example in Figure 7 shows the addition of "initTokens" to a transformation resource.

**MethodxForm**: This XML element is used to describe the ability of a transformation resource to transform an operand. As shown in Figure 1 and as reflected in the illustrative example in Figure 7, this element is nested within the Machine (i.e. transformation resources) element. This element is also represented as a transition in the Petri net. It thus requires the addition of attributes "gpsOffSetX" and "gpsOffSetY" to describe the locational GPS offset the transition will be displayed. It also requires the addition of the attribute "initTokens" to describe the number of tokens starting in the transition. Additionally it requires the attribute "dT" to describe the duration of time the process takes to complete. The illustrative example in Figure 7 shows the addition of "gpsOffSetX", "gpsOffSetY", "initTokens", and "dT" to transformation processes.

**MethodxPort**: As shown in Figure 1, the operations involving holding or transportation of an operand are universal to all of the physical resources. This element is also represented as a transition in the Petri net. It thus requires the addition of attributes "gpsOffSetX" and "gpsOffSetY" to describe the locational GPS offset the transition will be displayed. It also requires the addition of the attribute "initTokens" to describe the number of tokens starting in the transition. Additionally it requires the attribute "dT" to describe the duration of time the process takes to complete. The illustrative example in Figure 7 shows the addition of "gpsOffSetX", "gpsOffSetY", "initTokens", and "dT" to transportation processes.

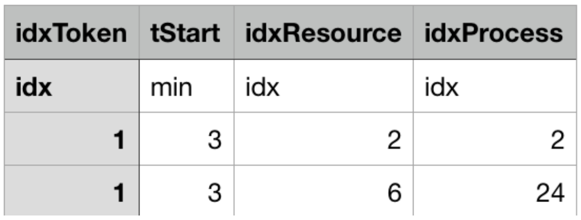


```xml
    <Machine name="House" gpsX="4" gpsY="4" controller="End User" initToken="0,0,1">
        <MethodxForm name="consume water" status="true" operand="electric power at 132kV, potable water" output="" gpsOffSetX="-1" gpsOffSetY="1" initToken="0" dT="1"/>
        <MethodxPort name="store" status="true" origin="House" dest="House" ref="park EV" operand="EV" output="EV" gpsOffSetX="1" gpsOffSetY="1" initToken="0" dT="1"/>
        <MethodxPort name="store" status="true" origin="House" dest="House" ref="charge EV" operand="EV, electric power at 132kV" output="EV" gpsOffSetX="1" gpsOffSetY="-1" initToken="0" dT="1"/>
    </Machine>
    <IndBuffer name="Work Location" gpsX="4" gpsY="0" controller="End User" initToken="0,0,0">
        <MethodxPort name="store" status="true" origin="Work Location" dest="Work Location" ref="park EV" operand="EV" output="EV" gpsOffSetX="1" gpsOffSetY="1" initToken="0" dT="1"/>
    </IndBuffer>
    <Transporter name="Water Pipeline" controller="Water Utility">
        <MethodxPort name="transport" status="true" origin="Water Treatment Facility" dest="House" ref="potable water" operand="potable water" output="potable water" gpsOffSetX="0" gpsOffSetY="0" initToken="0" dT="1"/>
    </Transporter>
```

Figure 7: An example HFGT Toolbox input XML input file that is PetriNet GUI compatible

## Creating a HFGT Petri Net Scheduled Event List Input File

The Scheduled Event List is required by the Petri net GUI to visualize the flow of (operand) tokens. It uses a CSV file format comprised of several variables where each row represents an event. An event is defined as the firing of a DOF on a specific token. Each event thus requires the following variables to be defined with Figure 8 providing an illustrative example of the scheduled event list.



| idxToken | tStart | idxResource | idxProcess |
|---|---|---|---|
| idx | min | idx | idx |
| 1 | 3 | 2 | 2 |
| 1 | 3 | 6 | 24 |

Figure 8: An example Scheduled Event List input CSV file that is required but the PetriNet GUI.

**idxToken**: This variable in the ScheduledEventList specifies the specific token being operated on by the DOF. This allows for the tracking of individual tokens to be tracked as they flow through the system.

**tStart**: This variable in the ScheduledEventList specifies the time at which the event begins.

**idxResource**: This variable in the ScheduledEventList specifies the index of the resource at which the event fires. The variable is later used to calculate the specific DOF being called by the event.

**idxProcess**: This variable in the ScheduledEventList specifies the index of the process that is used in the event. The variable is later used to calculate the specific DOF being called by the event.

## Petri net Data Structures and Functions

The *Petri Net Visualization* uses MATLAB's object-oriented programming functionality to create generic Petri net class structure. The primary purpose of the Petri net toolbox is to instantiate and subsequently populate the `myPetriNetwork` structure from the data in the `myLFES` structure and scheduled event list. It's associated class diagram is shown in Figure 9. The `myPetriNetwork` stores all of the information from the scheduled event list and required information from the `myLFES` structure. `myPetriNetwork` then calculates all of the mathematical quantities for tracking the flow of operand tokens. This `myPetriNetwork` object is filled in by the constructor

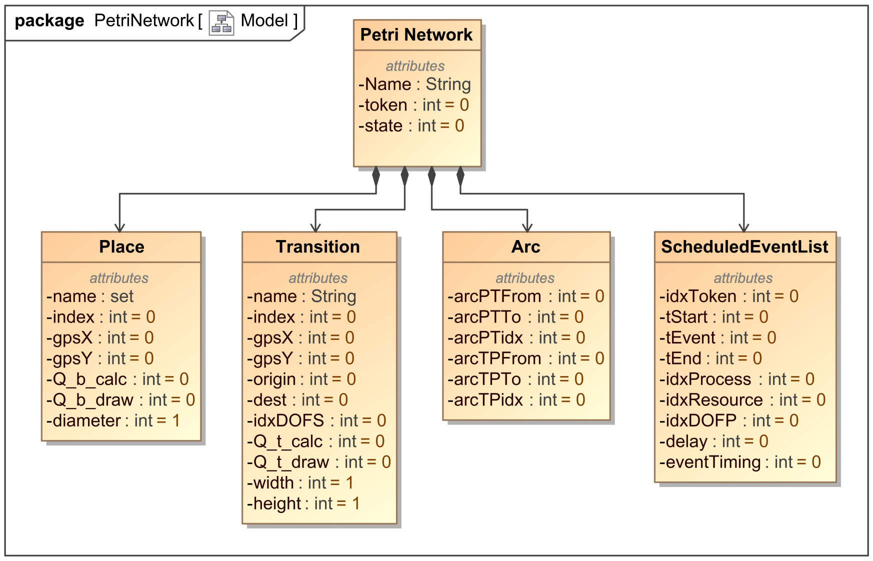


Figure 9: Structure of myLFES on completion of XML2LFES

method of PetriNetwork when given a `myLFES` object and a scheduled event List. Once constructed several methods belonging to myPetriNetwork are called by the GUI to calculate the matrices describing the flow of tokens and used for visualizing the discrete events. Their sequence is shown in Figure 10. In brief, the `import__()` methods serve to import the `myLFES` data and scheduled event list into their respective classes. Then, the `setInitQ()`, `populateQCalc()`, `makeQDraw()` methods calculate the matrices used to track the flow of tokens mathematically and visually. Finally, the `draw__()` methods create the graphical objects that are drawn on the GUI axis to visualize the Petri net. Each of these methods is called in turn by the GUI leaving `myPetriNetwork` in a "full" and drawn state. Each method is now discussed in detail.

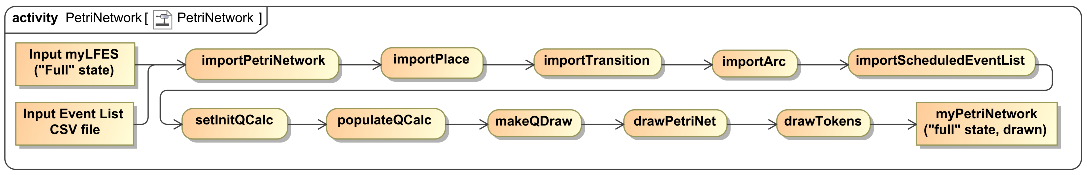


Figure 10: A high-level activity diagram of the Petri Net Toolbox

## myPetriNetwork = PetriNetwork(myLFES, scheduledEventList)

As previously mentioned, the `PetriNetwork()` constructor method serves to import the input myLFES and scheduled event list file and create the `myPetriNetwork` ($N$) data structure. This functionality is achieved through the sequence of functions shown in Figure 10.

**myPetriNetwork = importPetriNetwork(myLFES, scheduledEventList)** : This function is the PetriNetwork constructor and consequently instantiates `myPetriNetwork` ($N$). The high-level integer attributes of `myPetriNetwork` are initialized to zero and the sub-classes are initialized through their constructor methods.

**place = importPlace(myLFES)** : This function is the Place constructor and consequently instantiates
`myPetriNetwork.place` ($S$). This constructor assigns information to the sub-object `myPetriNetwork.place` ($S$) by "unpacking" `myLFES.machines` ($M$)and `myLFES.indBuffers` ($B$).

**transition = importTransition(myLFES)** : This function is the Transition constructor and consequently instantiates
`myPetriNetwork.transition` ($\mathcal{E}$). This constructor assigns information to the sub-object `myPetriNetwork.transition` ($\mathcal{E}$) by "unpacking" `myLFES.machines`$\{m\}$`.methodsxForm`, `myLFES.machines`$\{m\}$`.methodsxPort`,
`myLFES.indBuffers`$\{b\}$`.methodsxPort`, and `myLFES.transporters`$\{h\}$`.methodsxPort`.
All processes are equated to transitions in the Petri net.

**arc = importArc(myPetriNetwork)** : This function is the Arc constructor and consequently instantiates `myPetriNetwork.arc` ($M$). This constructor assigns information to the sub-object `myPetriNetwork.arc` ($M$) by "unpacking" the place's locations and the transition's origins and destinations. The associated arcs linking places and transitions as subsequently formed designating the direct of token flows.

**scheduledEventList = importScheduledEventList(eventList, myLFES)** : This function is the ScheduledEventList constructor and consequently instantiates `myPetriNetwork.scheduledEventList`. This constructor assigns information to the sub-object `myPetriNetwork.scheduledEventList` by "unpacking" the input `ScheduledEventList.CSV` and mapping events to their associated DOFs in `myLFES`.

**myPetriNetwork = setInitQCalc(myPetriNetwork)** : This function expands the size of the $Q_{b_{calc}}$ and $Q_{t_{calc}}$ matrices in the sub-objects `myPetriNetwork.place` ($S$) and `myPetriNetwork.transition` ($\mathcal{E}$) respectively such that they match the number of unique timings in the scheduled even list

**myPetriNetwork = populateQCalc(myPetriNetwork, myLFES)** : This function populates the $Q_{b_{calc}}$ ($Q_S$) and $Q_{t_{calc}}$ ($Q_E$) matrices in the sub-objects `myPetriNetwork.place` ($S$)and `myPetriNetwork.transition` ($\mathcal{E}$) respectively. Through tracking the activation of DOFs the flow operands through the system is tracked by the $Q_{b_{calc}}$ ($Q_S$) and $Q_{t_{calc}}$ ($Q_E$) matrices.

**myPetriNetwork = makeQDraw(myPetriNetwork)** : This function makes the $Q_{b_{draw}}$ and $Q_{t_{draw}}$ matrices in the sub-objects `myPetriNetwork.place` and `myPetriNetwork.transition` respectively. These matrices are used to draw the calculated amount of tokens in each drawn place and transition.

**myPetriNetwork = drawPetriNet(myPetriNetwork)** : This function creates the geometric objects that are drawn on the GUI main axis to visualize the places and transitions of the Petri net.

**newHandles = drawTokens(myPetriNetwork,number, color, handles)** : This function draws the tokens at each state of the Petri net. It uses information from the GUI stored in the handles structure and uses the `color` and `number` check boxes to determine if the number of tokens should be displayed via color map, numerically, or both.

## Petri net GUI Operation

The Petri net GUI uses the HFGT toolbox and Petri net structures to visualize the flow of operands through the HFGT system. The GUI is shown in Figure 11. The GUI components and operation are explained in turn.

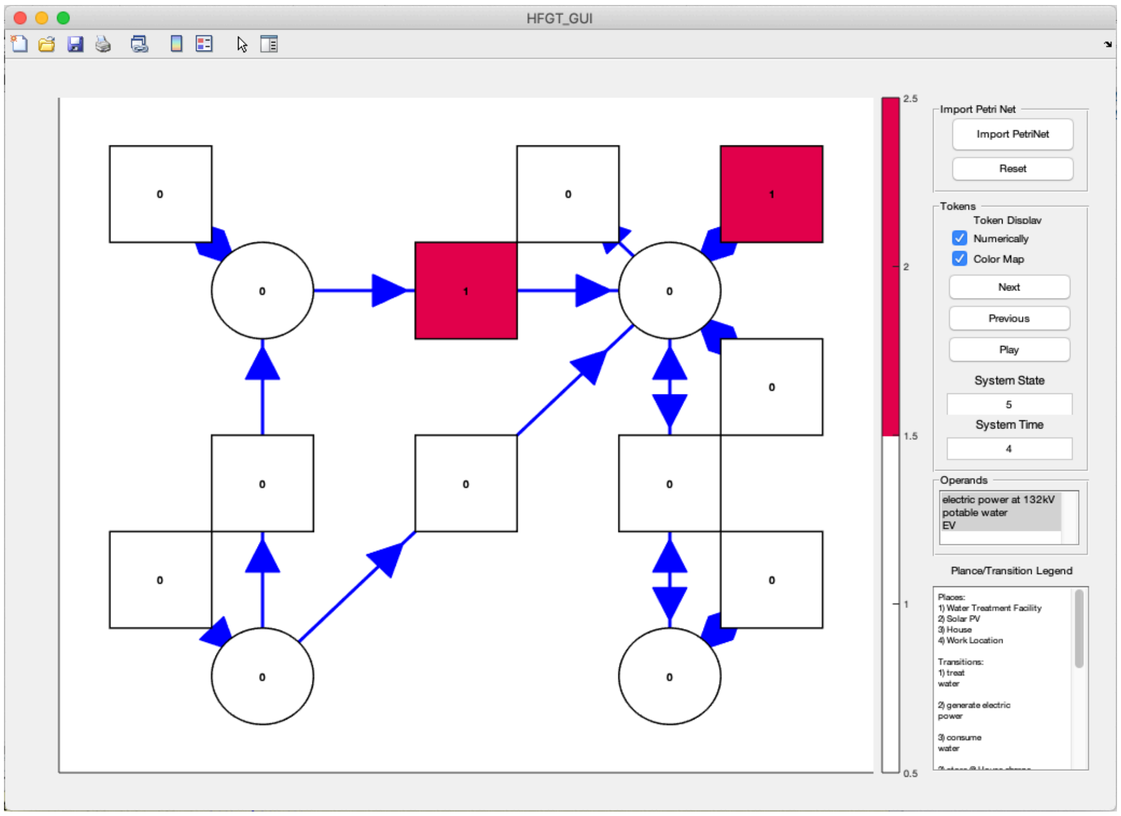
**initAxes, Axis** : This is the main axis on which the Petri Net is displayed when plotted. With the toolbar above it, the user is able to translate the plot and adjust the zoom of the plots view. Additionally, when displaying tokens with color, a vertical color bar will appear on the right side to define the color scale.

**Import PetriNet, Button** : This button is a method for importing the Petri Net onto the main axis through a selection menu using dropdown menus and browse buttons.

**Reset, Button** : This button removes any Petri Net that has been imported into the GUI and clears the main axis of any graphic displayed on it.

**Numerically, Check Box** : This check box designates whether or not to display tokens numerically within each Place and Transition.

**Color Map, Check Box** : This check box designates whether or not to display tokens with a colormap[51] normalized over Places and Transitions.



System State: 5

System Time: 4

✓ Numerically

✓ Color Map

electric power at 132kV
potable water
EV

Places:
1) Water Treatment Facility
2) Solar PV
3) House
4) Work Location

Transitions:
1) treat
water

2) generate electric
power

3) consume
water

Figure 11: An image of the Petri net GUI

**Next, Button** : This button advances the displayed Petri Net by incrementing the system state up one event.

**Previous, Button** : This button decrements the displayed Petri Net by decrementing the system state down one event.

**Play, Button** : This button continuously increments through the events of the displayed Petri Net. Through a selection menu, the animation speed can be adjusted along with the animation being saved as an mp4 video file.

**System State, Edit Text** : This edit text box displays the index of the current state depicted by the main axis. Through entering a valid event index into the edit text box, the main axis will then display the desired state.

**System Time, Edit Text** : This edit text box displays the time associated with the current state depicted by the main axis. Through entering a time into the edit text box, the main axis will then display the state associated with a time closest to that is desired.

**Operands, Select Text** : This select text box displays the operands present in the system. The main axis will only display tokens and transitions acting on the selected operands. Any combination of operands from one to all can be selected.

**Place/Transition Legend, Inactive Edit Text** : This inactive edit text box displays the index and name of each place and transition depicted on the main axis as '$IndexName$'.

## Conclusion

This paper serves as a user guide to a recently developed *Hetero-functional Graph Theory Toolbox* and its associated *Petri net Graphical User Interface*. The toolbox and GUI are written in the MATLAB[41] language and has been tested with v9.6 (R2019a). It is openly available on GitHub together with a sample input XML file for straightforward re-use. The paper details the syntax and semantics of the XML input, the myLFES (large flexible engineering system) data structure at the core of the toolbox and the functions used to construct and populate this data structure. The paper also details the syntax and structure of the input scheduled event list, the myPetriNetwork, and the operation of the GUI. The toolbox has been fully validated against several peer-review HFGT publications. The development of a streamlined, computationally efficient, and openly-accessible toolbox and GUI that automates and visualizes the underlying mathematical operations of HFGT enables the broader scientific community to apply HFGT to a wide variety of highly interconnected and heterogeneous engineering systems. Thus, this work enables several avenues for future research; particularly in the analysis, design, planning and operation of systems-of-systems.

## (2) Availability

### Operating system

The toolbox and GUI are compatible with macOS 10.12 and higher, Windows 7, Windows 10, and all other operating systems that support MATLAB[41] v9.6 (R2019a).

### Programming language

The toolbox and GUI are written in the MATLAB[41] language and has been tested with v9.6 (R2019a).

### Additional system requirements

The toolbox requires 1.4MB on disc with 8GB of ram recommended.

### Dependencies

The Hetero-functional Graph Theory Toolbox is dependent on a XML[45] input file structure. The Hetero-Functional Graph Theory Toolbox also utilizes sparse tensors in its calculations and thus is dependent on the Tensor Toolbox for Matlab[46, 47].

### List of contributors

Dakota J. Thompson is the lead author on this paper, secondary developer of the hetero-functional graph theory, and lead developer of the Petri net GUI. Prabhat Hegde contributed to the writing this paper and contributed to the development of the hetero-functional graph theory toolbox. Wester C.H. Schoonenberg is the lead developer of the hetero-functional graph theory toolbox and contributed to the writing of this paper. Inas S. Khayal architected and managed the Petri net GUI and contributed to the development of hetero-functional graph theory. Amro M. Farid architected the hetero-functional graph theory toolbox and Petri net GUI and managed the project including the writing of this paper.

**Software location:**

**Code repository**

**Name:** GitHub
**Persistent identifier:** https://github.com/LIINES/HFGTToolbox
**Licence:** tbc
**Date published:** 28/08/20

## Language

The toolbox and GUI are written in the MATLAB[41] language and has been tested with v9.6 (R2019a).

## (3) Reuse potential

The development of a streamlined, computationally efficient, and openly-accessible toolbox and GUI that automates and visualizes the underlying mathematical operations of HFGT enables the broader scientific community to apply HFGT to a wide variety of highly interconnected and heterogeneous engineering systems. Thus, this work enables several avenues for future research; particularly in the analysis, design, planning and operation of systems-of-systems. The GitHub repository will be updated appropriately as the software is modified or extended to support the work.

## Competing interests

The authors declare that they have no competing interests.

## References

[1] Olivier L De Weck, Daniel Roos, and Christopher L Magee. *Engineering systems: meeting human needs in a complex technological world*. MIT Press, Cambridge, Mass., 2011.

[2] Amro M. Farid. An engineering systems introduction to axiomatic design. In Amro M. Farid and Nam P Suh, editors, *Axiomatic Design in Large Systems: Complex Products, Buildings & Manufacturing Systems*, chapter 1, pages 1–47. Springer, Berlin, Heidelberg, 2016.

[3] SE Handbook Working Group. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. International Council on Systems Engineering (INCOSE), 2015.

[4] J Rumbaugh, I Jacobson, and G Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Mass., 2005.

[5] Tim. Weilkiens. *Systems engineering with SysML/UML modeling, analysis, design*. Morgan Kaufmann, Burlington, Mass., 2007.

[6] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, Burlington, MA, 2nd edition, 2011.

[7] Edward Crawley, Bruce Cameron, and Daniel Selva. *System Architecture: Strategy and Product Development for Complex Systems*. Prentice Hall Press, Upper Saddle River, N.J., 2015.

[8] Mark Newman, Albert-Laszlo Barabasi, and Duncan J Watts. *The structure and dynamics of networks*. Princeton University Press, 2006.

[9] Albert-László Barabási et al. *Network science*. Cambridge university press, 2016.

[10] Feng Xie and David Levinson. Modeling the growth of transportation networks: A comprehensive review. *Networks and Spatial Economics*, 9(3):291–307, 2009.

[11] Réka Albert, István Albert, and Gary L Nakarado. Structural vulnerability of the north american power grid. *Physical review E*, 69(2):025103, 2004.

[12] Luıs A Nunes Amaral, Antonio Scala, Marc Barthelemy, and H Eugene Stanley. Classes of small-world networks. *Proceedings of the national academy of sciences*, 97(21):11149–11152, 2000.

[13] Armando Di Nardo, Michele di natale, Giovanni Santonastaso, Velitchko Tzatchkov, and Víctor Yamanaka. Water network sectorization based on graph theory and energy performance indices. *ASCE Journal of Water Resources Planning and Management*, 140:–, 01 2013.

[14] Dmitry Ivanov, Alexandre Dolgui, and Boris Sokolov. Applicability of optimal control theory to adaptive supply chain planning and scheduling. *Annual Reviews in control*, 36(1):73–84, 2012.

[15] Yuehwern Yih. *Handbook of healthcare delivery systems*. CRC Press, Boca Raton, FL, 2016.

[16] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. Multilayer networks. *Journal of complex networks*, 2(3):203–271, 2014.

[17] Wester C.H. Schoonenberg, Inas S. Khayal, and Amro M. Farid. *A Heterofunctional Graph Theory for Modeling Interdependent Smart City Infrastructure*. Springer, Berlin, Heidelberg, 2018.

[18] Steffi O. Muhanji, Alison E. Flint, and Amro M. Farid. *eIoT: The Development of the Energy Internet of Things in Energy Infrastructure*. Springer, Berlin, Heidelberg, 2019.

[19] D McFarlane, S Sarma, J L Chirn, K Ashton, and C Y Wong. Auto ID systems and intelligent manufacturing control. *Engineering Applications of Artificial Intelligence*, 16(4):365–376, 2003.

[20] Duncan Mcfarlane, Vivek Agarwal, Alia Ahmad Zaharudin, Chien Yaw Wong, Robin Koh, and Yun Kang. The Intelligent Product Driven Supply Chain. In *2002 IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 393–398, 2002.

[21] Duncan McFarlane, Vaggelis Giannikas, Alex CY Wong, and Mark Harrison. Product intelligence in industrial control: Theory and practice. *Annual Reviews in Control*, 37(1):69–88, 2013.

[22] Gerben G Meyer, Kary Främling, and Jan Holmström. Intelligent products: A survey. *Computers in Industry*, 60(3):137–148, 2009.

[23] Amro M. Farid and Duncan C McFarlane. A Development of Degrees of Freedom for Manufacturing Systems. In *IMS'2006: 5th International Symposium on Intelligent Manufacturing Systems: Agents and Virtual Worlds*, pages 1–6, Sakarya, Turkey, 2006.

[24] A M Farid. *Reconfigurability Measurement in Automated Manufacturing Systems*. Ph.d. dissertation, University of Cambridge Engineering Department Institute for Manufacturing, 2007.

[25] Amro M. Farid. Product Degrees of Freedom as Manufacturing System Reconfiguration Potential Measures. *International Transactions on Systems Science and Applications – invited paper*, 4(3):227–242, 2008.

[26] A M Farid and D C McFarlane. Production degrees of freedom as manufacturing system reconfiguration potential measures. *Proceedings of the Institution of Mechanical Engineers, Part B (Journal of Engineering Manufacture) – invited paper*, 222(B10):1301–1314, 2008.

[27] Amro M Farid. Multi-Agent System Design Principles for Resilient Coordination and Control of Future Power Systems. *Intelligent Industrial Systems*, 1(3):255–269, 2015.

[28] Dakota Thompson, Wester C.H. Schoonenberg, and Amro M. Farid. A Hetero-functional Graph Analysis of Electric Power System Structural Resilience. In *IEEE Innovative Smart Grid Technologies Conference North America*, pages 1–5, Washington, DC, United states, 2020.

[29] William Naggaga Lubega and Amro M Farid. A Reference System Architecture for the Energy-Water Nexus. *IEEE Systems Journal*, PP(99):1–11, 2014.

[30] Catherine Carey and Amro M. Farid. The energy-water nexus: A hetero-functional graph theory centrality analysis. Technical report, Thayer School of Engineering at Dartmouth, 2018.

[31] Dakota Thompson and Amro M. Farid. Energy-water nexus: Reference architecture, hetero-functional graph theory analysis and equations of motion. Technical report, Thayer School of Engineering at Dartmouth, 2019.

[32] Amro M. Farid. Symmetrica: Test Case for Transportation Electrification Research. *Infrastructure Complexity*, 2(9):1–10, 2015.

[33] Amro M. Farid. A Hybrid Dynamic System Model for Multi-Modal Transportation Electrification. *IEEE Transactions on Control System Technology*, PP(99):1–12, 2016.

[34] Thomas JT van der Wardt and Amro M Farid. A hybrid dynamic system assessment methodology for multi-modal transportation-electrification. *Energies*, 10(5):653, 2017.

[35] Wester C.H. Schoonenberg and Amro M. Farid. A Dynamic Model for the Energy Management of Microgrid-Enabled Production Systems. *Journal of Cleaner Production*, 1(1):1–10, 2017.

[36] Wester C.H. Schoonenberg and Amro M. Farid. A dynamic production model for industrial systems energy management. In *2015 IEEE International Conference on Systems Man and Cybernetics*, pages 1–7, Hong Kong, 2015.

[37] Inas S. Khayal and Amro M. Farid. Architecting a System Model for Personalized Healthcare Delivery and Managed Individual Health Outcomes. *Complexity*, 1(1):1–25, 2018.

[38] Inas S. Khayal and Amro M. Farid. The Need for Systems Tools in the Practice of Clinical Medicine. *Systems Engineering*, 20(1):3–20, 2016.

[39] Inas S. Khayal and Amro M. Farid. A Dynamic System Model for Personalized Healthcare Delivery and Managed Individual Health Outcomes. *submitted to: Health Systems*, 1(1):1–18, 2020.

[40] Prabhat Hegde, Wester C.H. Schoonenberg, Dakota Thompson, and Amro M. Farid. The hetero-functional graph theory toolbox. Technical report, Laboratory for Intelligent Integrated Networks of Engineering Systems. Thayer School of Engineering at Dartmouth, San Francisco, CA, United states, 2020.

[41] MATLAB. *version 9.6.0 (R2019a)*. The MathWorks Inc., Natick, Massachusetts, 2019.

[42] T Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[43] Kurt Jensen. Coloured petri nets, volume 1: Basic concepts. *EATCS Monographs on Theoretical Computer Science. Springer-Verlag*, 1992.

[44] Claude Girault and Rüdiger Valk. *Petri nets for systems engineering: a guide to modeling, verification, and applications*. Springer Science & Business Media, 2013.

[45] Elliotte Harold and Scott Means. *XML in a nutshell*. O'Reilly Media, 01 2002.

[46] Brett W. Bader and Tamara G. Kolda. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, December 2007.

[47] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 3.1. Available online, June 2019.

[48] Tadao Murata. Temporal uncertainty and fuzzy-timing high-level petri nets. In Jonathan Billington and Wolfgang Reisig, editors, *Application and Theory of Petri Nets 1996*, volume 1091 of *Lecture Notes in Computer Science*, pages 11–28. Springer Berlin Heidelberg, 1996.

[49] Woulter Falkena. *xml2struct*. MATLAB Central File Exchange., 2020.

[50] Halima Abdulla and Amro M. Farid. Extending the energy-water nexus reference architecture to the sustainable development of agriculture, industry & commerce. In *First IEEE International Smart Cities Conference*, pages 1–7, Guadalajara, Mexico, 2015.

[51] Kenneth Moreland. Diverging color maps for scientific visualization. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Yoshinori Kuno, Junxian Wang, Renato Pajarola, Peter Lindstrom, André Hinkenjann, Miguel L. Encarnação, Cláudio T. Silva, and Daniel Coming, editors, *Advances in Visual Computing*, pages 92–103, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.