Azad M. Madni · Barry Boehm ·
Daniel Erwin · Mahta Moghaddam ·
Michael Sievers ·
Marilee Wheaton *Editors*

Recent Trends and Advances in Model Based Systems Engineering

Springer

Recent Trends and Advances in Model Based Systems Engineering

Azad M. Madni • Barry Boehm • Daniel Erwin
Mahta Moghaddam • Michael Sievers
Marilee Wheaton
Editors

Recent Trends and Advances in Model Based Systems Engineering

Springer

Editors
Azad M. Madni
Department of Astronautical Engineering
University of Southern California
Los Angeles, CA, USA

Daniel Erwin
Department of Astronautical Engineering
University of Southern California
Los Angeles, CA, USA

Michael Sievers
Department of Astronautical Engineering
University of Southern California
Los Angeles, CA, USA

Barry Boehm
Department of Industrial and Systems Engineering
University of Southern California
Los Angeles, CA, USA

Mahta Moghaddam
Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, CA, USA

Marilee Wheaton
Department of Astronautical Engineering
University of Southern California
Los Angeles, CA, USA

ISBN 978-3-030-82082-4 ISBN 978-3-030-82083-1 (eBook)
https://doi.org/10.1007/978-3-030-82083-1

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Systems engineering is undergoing an exciting transformation that is motivated by mission and system complexity and paced by advances in model-based systems engineering (MBSE) and digital engineering (DE). This transformation is enabled by Industry 4.0, the Internet of Things (IoT), and the ongoing convergence of systems engineering with other disciplines. The central theme of the 2020 Conference on Systems Engineering Research (CSER) was motivated by these developments. Specifically, this conference was focused on exploring recent trends and advances in model-based systems engineering (MBSE) and the synergy of MBSE with other disciplines (e.g., digital engineering) and technologies (e.g., simulation, AI, machine learning).

Systems engineering today has two major thrusts: traditional methods that work well for relatively mature and not overly complicated or complex systems, and new and innovative methods that are specifically driven by the increasing complexity of sociotechnical systems and advances in engineering, materials, computation, and convergence. The latter has become increasingly important for addressing problems in the twenty-first century. MBSE is rapidly becoming a vital system engineering advance to address such problems.

Researchers from academia, industry, and government submitted papers on a variety of MBSE topics for this conference. These include ontologies and MBSE, MBSE processes, model-based methods in systems architecting, modeling approaches in MBSE, MBSE standards, MBSE languages, synergy between MBSE and digital engineering (DE), economic analysis of MBSE, MBSE application areas (e.g., manufacturing, aerospace, defense), and the future of MBSE.

This volume is a compendium of peer-reviewed research papers from university, government, and industry researchers who participated in 2020 CSER. It brings together diverse domains and technical competencies of model-based systems engineering (MBSE) in a single, comprehensive volume. To help the reader conveniently navigate this volume, the chapters are organized into seven parts. Each part represents a key MBSE research area.

It is our hope that this volume will get the readers interested in pursuing MBSE research beyond the traditional application areas and take on complex scientific and

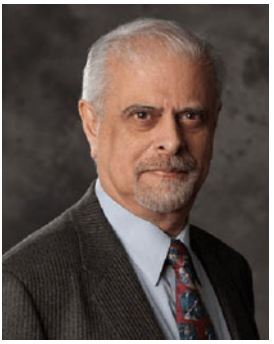
societal problems of national and global significance. On behalf of the editors, I want to thank all who contributed to this volume. We hope that you find the contents of this volume inspiring and potentially useful building blocks for future research.

University of Southern California, Azad M. Madni
Los Angeles, CA, USA

Contents

Part VI MBSE Applications

About the Editors



Azad M. Madni is a member of the National Academy of Engineering and the University Professor of Astronautics, Aerospace and Mechanical Engineering. He also has a joint appointment in Civil and Environmental Engineering at the University of Southern California's Viterbi School of Engineering. He is the holder of the Northrop Grumman Foundation Fred O'Green Chair in Engineering. He is the executive director of USC's systems architecting and engineering program and the founding director of the Distributed Autonomy and Intelligent Systems Laboratory. He is the founder and CEO of Intelligent Systems Technology, Inc., a hi-tech company that conducts research and offers educational courses in intelligent systems for education and training. He is the chief systems engineering advisor to The Aerospace Corporation. He received his Ph.D., M.S., and B.S. degrees in Engineering from UCLA. His recent awards include *2021 INCOSE Benefactor Award, 2021 IEEE AESS Judith A. Resnik Space Award, 2020 IEEE SMC Norbert Wiener Award, 2020 NDIA's Ferguson Award* for Excellence in Systems Engineering, *2020 IEEE-USA Entrepreneur Achievement Award, 2019 IEEE AESS Pioneer Award*, *2019 INCOSE Founders Award, 2019 AIAA/ASEE Leland Atwood Award*, *2019 ASME CIE Leadership Award, 2019 Society for Modeling and Simulation International Presidential Award, and 2011 INCOSE Pioneer Award.* He is a Life Fellow/Fellow of IEEE, INCOSE, AIAA, AAAS, SDPS, IETE, AAIA, and WAS. He

is the co-founder and current chair of IEEE SMC Technical Committee on Model Based Systems Engineering. He is the author of *Transdisciplinary Systems Engineering: Exploiting Convergence in a Hyper-Connected World* (Springer 2018). He is the co-author of *Tradeoff Decisions in System Design* (Springer, 2016).

Barry Boehm is a member of the National Academy of Engineering and Distinguished Professor of Computer Science in the Viterbi School of Engineering at the University of Southern California. His honors and awards include INCOSE Pioneer, guest lecturer at the USSR Academy of Sciences, the AIAA Information Systems Award, the J.D. Warnier Prize for Excellence in Information Sciences, the ISPA Freiman Award for Parametric Analysis, the NSIA Grace Murray Hopper Award, the Office of the Secretary of Defense Award for Excellence, the ASQC Lifetime Achievement Award, and the ACM Distinguished Research Award in Software Engineering . He is an AIAA Fellow, an ACM Fellow, an IEEE Fellow. He received his B.A. degree from Harvard University, and his M.S. and Ph.D. degrees from UCLA, all in mathematics.

Daniel Erwin is a professor and chair of astronautical engineering, and an associate fellow of AIAA. He is also the faculty member who oversaw the all-student research team that set the student altitude record for rockets to pass the Karman line into outer space. He is the co-director of USC's Distributed Autonomy and Intelligent Systems Laboratory. His awards include the 2017 Engineers' Council Distinguished Engineering Project Achievement Award, 2016 Engineers' Council Distinguished Engineering Educator Award, 2006 USC-LDS Student Association Outstanding Teaching Award, 1995 USC School of Engineering Outstanding Teaching Award, and 1993 TRW, Inc. TRW Excellence in Teaching Award. He received his B.S. in applied physics from California Institute of Technology and his M.S. and Ph.D. in electrical engineering from the University of Southern California.

Mahta Moghaddam is Distinguished Professor of Electrical and Computer Engineering and a member of the National Academy of Engineering. She is an IEEE Fellow and a past president of IEEE Antennas and Propagation Society. She is the editor-in-chief of the *IEEE Antennas and Propagation* journal. She has received several awards and honors including NASA Honor Award, Outstanding Public Service Leadership Medal, and numerous group achievement awards. She is the president and co-founder of Maxwell Medical Corporation, a USC startup that is developing medical diagnostic therapy and interoperative monitoring devices using microwave technology. She received her B.S. in electrical engineering from the University of Kansas with highest distinction, and her M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana-Champaign.

Marilee Wheaton is a systems engineering fellow of The Aerospace Corporation and president of the International Council on Systems Engineering. She is a fellow of AIAA and INCOSE, and a senior member of IEEE. Previously, she was a general manager at The Aerospace Corporation. She has also served as an adjunct associate professor of systems architecting and engineering program at the University of Southern California for 11 years. Marilee is a life member and fellow of the Society of Women Engineers. She is an advisory board member of California State University, Northridge College of Engineering and Computer Science. She received her B.A. degree in mathematics from California Lutheran University and her M.S. degree in systems engineering from University of Southern California. Marilee has been involved in the Conference on Systems Engineering Research in a leadership role for nearly a decade. She is currently pursuing her doctorate at the University of Southern California in astronautical engineering with a specialization in systems engineering.

Michael Sievers is a senior systems engineer at Caltech's Jet Propulsion Laboratory in Pasadena, California, and is responsible for developing and analyzing spacecraft and ground systems architectures. He is also an adjunct lecturer at the University of Southern California where he teaches classes in systems and system of systems architectures, engineered resilience, and model-based systems engineering. Dr. Sievers earned his Ph.D. and master's degrees in computer science and a Bachelors' degree in electrical engineering all from UCLA. He has published over 50 journal and conference papers and is an INCOSE Fellow, AIAA Associate Fellow, and IEEE Senior Member.

Part I
MBSE and Digital Engineering

Toward a Reference Architecture for Digital and Model-Based Engineering Information Systems

Hayden C. Daly and Paul T. Grogan

Abstract Digital and model-based engineering envisions a future where software systems are intricately involved in systems engineering and engineering design efforts. Recent advances in the field of software engineering have the potential to enable more flexible, reconfigurable, and updateable systems for engineering applications. This paper introduces an information system reference architecture for digital and model-based engineering activities based on modern web-based architectural styles. An application case explains how the reference architecture shaped the implementation of the Tradespace Analysis Tool for Constellations (TAT-C) Knowledge Base, a software component for space systems engineering that maintains a resource library conforming to common object schemas. Database, back-end, and front-end software components serve as architectural layers connected by simple information protocols based on semantic linked data models for improved interoperability.

Keywords Digital engineering · Layered architecture · Model-based engineering · Model interoperability · Semantic web technology · Software architecture

1 Introduction

Digital and model-based engineering (DMbE) envisions the widespread use of digital artifacts, digital environments, and digital tools to support engineering activities (Hale et al. 2017). It encompasses recent efforts in model-based systems engineering to adopt semantic frameworks and graphical modeling languages as a means to represent systems models in a common, interoperable format (Bone et al. 2018, 2019). However, developing an infrastructure platform for information technology that is "flexible, reconfigurable, and updateable" remains a significant

H. C. Daly · P. T. Grogan (✉)
Castle Point on Hudson, Stevens Institute of Technology, Hoboken, NJ, USA

A. M. Madni et al. (eds.), *Recent Trends and Advances in Model Based Systems Engineering*, https://doi.org/10.1007/978-3-030-82083-1_1

challenge in DMbE (Hale et al. 2017). This challenge sits at the intersection between two fields that increasingly overlap: systems engineering and software engineering.

The field of software engineering has experienced significant growth, innovation, and change in recent decades. Indeed, many of the systems modeling languages including SysML, OPM, and IDEF0 evolved from software engineering practice established in the 1990s to standardize software design and use object-oriented programming styles to accommodate greater levels of product complexity (Dori 2016). More recent trends in software engineering focus on service-orientation, web-based application programming interfaces (APIs), and containerization as further systems-level techniques to accommodate increased product complexity with more distributed software architectures.

Drawing from the state-of-the-art in software engineering practice, this paper advances a reference software architecture to support DMbE practices. The proposed reference architecture has been successfully implemented for the Tradespace Analysis Tool for Constellations Knowledge Base (TAT-C KB), a systems engineering software tool in the domain of space systems. Based on insights and experience from this application case, the reference architecture has the potential to serve as the foundation for future DMbE information systems.

2 Background

The term *reference architecture* originated from the field of software engineering but, over the past decade, has been adopted in systems engineering to describe "the essence of existing architectures, and the vision of future needs and evolution to provide guidance to assist in developing new system architectures" (Cloutier et al. 2010). From a system design perspective, a reference architecture encodes patterns or rules and "constrains the instantiations of multiple architectures and solutions" to provide a foundation and comparison point for individual solutions (Office of the Assistant Secretary of Defense 2010).

Large-scale software systems such as the Internet follow a layered architecture as a form of modularity enabling robustness (disturbances do not easily propagate across layers) while also preserving changeability (component layers can be updated in relative isolation) (Doyle et al. 2011). In the case of the Internet, various layers encapsulate the data link, network, transport, session, and presentation components. Each component layer is constrained by a set of protocols; however, shared constraints across layers deconstrain the overall system by allowing piece-wise substitution and evolution.

Software development practice increasingly emphasizes layers at the individual application level to achieve similar lifecycle objectives. Architectural styles such as representational state transfer (REST) constrain protocols to stateless resource operations to minimize latency and maximize independence and scalability (Fielding and Taylor 2002). Improved access to customize server-side components using common scripting languages such as Python (Flask) and JavaScript (Node.js/Express)

and availability of real-time, bidirectional, client–server communication libraries (WebSockets and Socket.IO) support layered architectures even for small-scale applications.

Despite recent advances in software engineering, new architectural styles are slow to translate to DMbE environments which still emphasize centralized platforms for aggregating graphical models and proprietary software without exposed APIs. A vision for a future DMbE information system resembles that of the Internet where component models are orchestrated in layers with well-defined constraining protocols. Drawing from recent work in the space systems engineering domain, this paper outlines a reference architecture to explain how a modern web-based layered architecture can support DMbE activities. The reference architecture highlights the key components (layers) and protocols to exchange information.

3 Proposed Reference Architecture

3.1 System Components

The proposed software architecture consists of a back-end, front-end, and database. The database component can be any data storage solution and could be implemented in different ways. It could be implemented as a file system, relational database, non-relational database, and in-memory store in this architecture. The database solution can be respective to the data used in the application, and all that really matters is that it will be able to communicate directly with the back-end.

The back-end connects the database and front-end/client. The primary purpose of the back-end is to convert the database command line interface into an easily accessible HTTP (HyperText Transfer Protocol) service. The back-end acts as an API that communicates with the front-end and user through the same communication strategy. This API is a wrapper for the underlying technical models and provides analysis services. This back-end can take the form of various technologies such as Flask, Node.js/Express, Apache, Nginx, and more as long as the solution allows for HTTP accessible services.

The front-end can be implemented by choice and is application specific. The main function of this component is to allow to directly interact with the service. The front-end can take many forms such as a browser-based GUI (Graphical User Interface), a mobile application, or direct communication with the client. The proposed communication strategy between these components is HTTP requests as they are a common and straightforward communication method utilized within web technology. The benefits of this communication strategy will be stressed in the following section.

As shown in Fig. 1, the proposed architecture consists of three simple components with communication only done through HTTP. The back-end can communicate with the database in whatever method applicable to the database strategy/usage. Many applications of this architecture could have three entirely isolated components

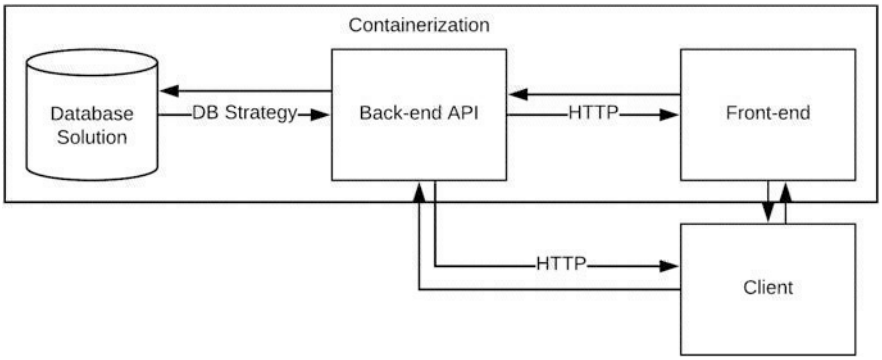


Fig. 1 UML diagram of reference architecture

or just two if the back-end and the database are stored in the same component. The entire application can be containerized using recently popular technologies like Docker and Kubernetes.

3.2 System Interface

Communication through HTTP requests is beneficial for numerous reasons, but the primary three are to reduce development work, reduce integration challenges, and improve overall operation. HTTP is the Internet standard for communication and involves the transfer of data over a URL (Uniform Resource Locator) request. These requests can contain data in the form of JSON (JavaScript Object Notation), XML (Extensible Markup Language), or other types. There are a few different fundamental forms of HTTP requests utilized in this software application which are GET, POST, and DELETE. A GET request has a specified URL and retrieves information from the server. A POST request sends information to the server. A DELETE request is meant to delete specified information from the server.

By utilizing HTTP for communication, the API has direct communication with not only the front-end but the user as well leading to greater interoperability. This reduces the development work by allowing the developer to fixate their responsibility solely on building a functional API and not the logistics of how it would communicate with the front-end. This standardized communication reduces integration difficulties for the same reason. The engineers can work with a standard interface of the HTTP requests rather than worry about developing an API and communication strategy. This leads to an understanding of a standard model for communication with support of numerous web technologies already. Lastly, it improves the overall operation because the front-end and back-end run independent of one another so their individual performances will not have an effect on one another.

3.3 Data Encoding

By utilizing communication through HTTP, the model also has to choose what syntax to format data in. JSON is a popular format for HTTP data transfer that uses a key-value (dictionary) structure to encode data. This type of data is very flexible for engineers to use as it does not require typing for any of the object properties and allows for appending of extra fields. The JSON format is also very easy for engineers to use as it is human readable and does not require much additional training.

Despite JSON's flexibility, it can still follow schema specifications. This can be achieved through the use of JSON-LD which provides standard guidelines for JSON communication (Sporny et al. 2019). Using the JSON-LD format requires the implementation of standardized schema of the data. A schema includes simple documentation about what should be sent and what variables fields represent including units, reference, and other parameters. This stresses the concept of the semantic web where everything can be in communication in a way that is easily accessible and has consistent semantics. Another major benefit of the schema is that enables interoperability based on common understanding. With the recent applications of machine learning in the field of systems engineering to discover insights on data, the usage of a standardized data will lead to much more ease on the application of processing.

For usage of standardized schema, resources such as Schema.org can be utilized (Guha et al. 2016). Schema.org is a database of schema for structured data on the Internet with the overarching goal of facilitating the semantic web.

4 Example Application Case

4.1 Tradespace Analysis Tool for Constellations (TAT-C)

The Tradespace Analysis Tool for Constellations (TAT-C) is a software modeling tool to support pre-Phase A conceptual design of Earth-observing spacecraft constellations (Le Moigne et al. 2017). Based on a mission concept and constraints on available constellation geometries, spacecraft buses, and instruments, TAT-C enumerates and searches a combinatorial tradespace to identify desirable mission architectures. Software modules in TAT-C perform specific functions such as orbital propagation, launch vehicle selection, instrument performance analysis, cost analysis, and search execution.

The TAT-C Knowledge Base (KB) module documents schema definitions for TAT-C objects and maintains a library of conforming object models gathered from historical missions. Designed as a layered architecture with database, back-end, and front-end components, other TAT-C components including a browser-based GUI access KB data resources using standard web-based protocols. A publicly accessible version of the KB application is available at https://tatckb.org.

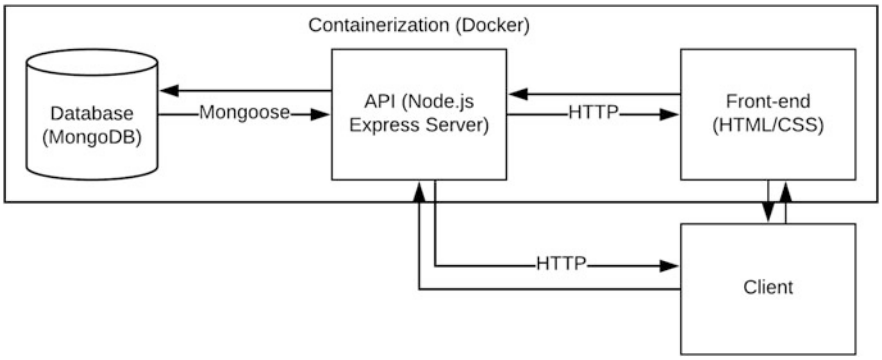


Fig. 2 UML diagram of architecture for TAT-C

Although small in scale, TAT-C exhibits many of the challenges of DMbE information systems. The KB implementation details in the following sections address the implementation of individual components, protocols exchanging information between layers, and the data encoding system selected to improve model interoperability between software components.

4.2 System Components

The TAT-C KB software architecture consists of a front-end website, a back-end API, and a database. The decision was made to isolate the front-end code from the back-end code and came into fruition as a front-end GUI and a back-end server only in communication through HTTP requests. Figure 2 shows the KB architecture.

The front-end component uses common web technology and takes the form of an HTML/CSS/JavaScript website to act as an interface for the API. The front-end interface can be seen in Fig. 3. The back-end component uses a Node.js/Express server to act as wrapper for the database and provide technical data analysis. The back-end uses Mongoose to make queries from the MongoDB database.

Containerization was also utilized on this project as it reduced friction with integration and will be expanded upon in Sect. 4.5.

4.3 System Interface

Communication between the components relies on HTTP requests. As shown in Fig. 4, the client has the choice of either interacting with the front-end GUI or making a request from the API directly via HTTP. For this function specifically, the only parameter the API requires is the collection being requested. The API

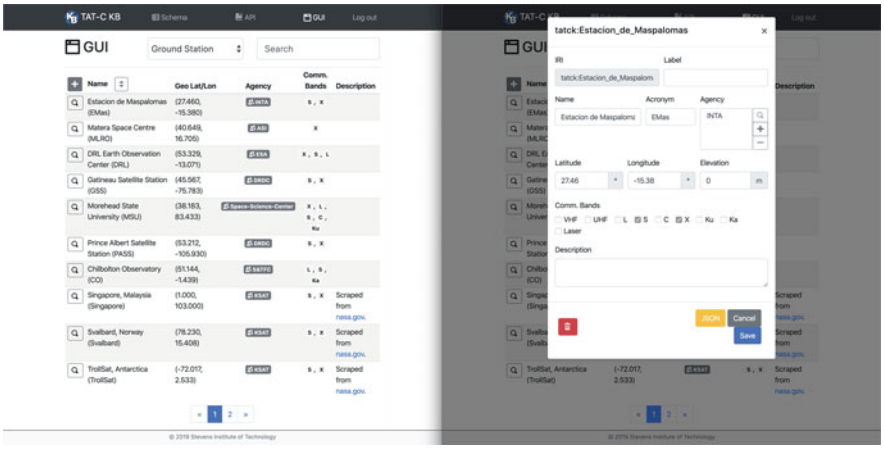


Fig. 3 The front-end interface of TAT-C KB

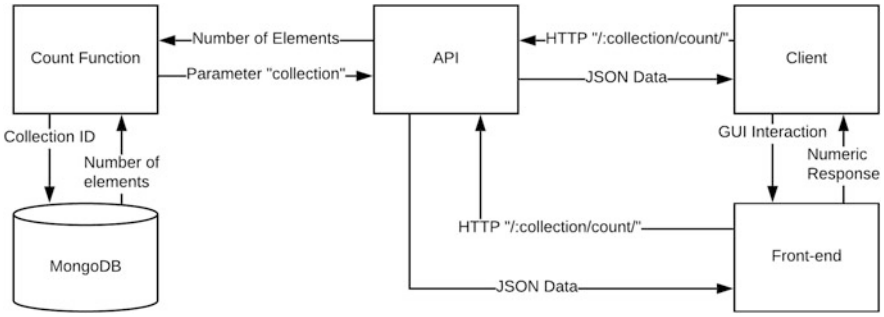


Fig. 4 Communication diagram for count HTTP requests

then redirects their request to the internal count function with the parameter of the collection and returns the number of elements in the specified collection. All the count function does is query the database for the number in that collection.

The communication with the API allowed for a few different kinds of HTTP requests which were: `count`, `list`, `get`, `add`, and `delete`. All of these have specific parameters and queries. The parameters are required fields by the API for the function. The arguments are additional fields the user can use when constructing API requests to get certain responses. Table 1 documents all of the parameters and queries which summarizes the API documentation.

The `count` function is pretty straightforward and provides the number of a specified type within the database. This allows for the additional query `search` where you are able to find the number of objects that match a specified string within the type. The `list` function provides a list of all the objects within a specified type and allows for the following queries: search, sort, offset, populate, and limit. The `get` function is the simplest and just allows you to get an object of a specified type

Table 1 All HTTP requests

Request	Method	URL	Parameters	Arguments
Count	GET	`/:type:/count/`	`type`	`search`
List	GET	`/:type/list/`	`type`	`search`, `sort`, `offset`, `populate`, `limit`
Get	GET	`/:type/:id/` `/:type/instance/:id`	`type`, `id`	`populate`
Add	POST	`/:type/add/` `/add/`	`type`	`token`
Delete	DELETE	`/:type/delete/:id/`	`type`, `id`	`token`


```
{
    @context: {
      owl: "http://www.w3.org/2002/07/owl#",
      rdf: "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
      rdfs: "http://www.w3.org/2000/01/rdf-schema#",
      schema: "http://schema.org/",
      tatckb: "http://tatckb.org/schema/2.0/"
    },
    @graph: [
        {
          @id: "tatckb:ASI",
          @type: "tatckb:Agency",
          tatckb:name: "Agenzia Spaziale Italiana",
          tatckb:acronym: "ASI",
          tatckb:agencyType: "GOVERNMENT",
          @lastUpdated: "2019-07-24T20:10:06.558Z"
        }
    ]
}
```


Fig. 5 Output of "/api/agency/list?search=ASI"

and `@id` field. This request only allows the query `populate` which will populate all sub-objects by the `@id` field within the object. The `add` function takes the input of an object and allows you to add it to a specified type. The delete function allows you to delete an object within a specified type by the `@id` field within the object and accepts no additional queries. An example of the JSON response from the API is in Fig. 5 and will be explained in the following section.

4.4 Data Encoding

The overall goal of the KB is to set a standard for tradespace analysis data and help organize it. To increase interoperability, specific JSON-LD schemas were created for

Table 2 Schema documentation for data type `GroundStation`

Property	Expected type	Description
`name`	`schema:Text`	The full name of an entity
`latitude`	`schema:Number`	Latitude (decimal degrees) with respect to the WGS 84 geodetic model. Ranges between −90° (south) and 90° (north) where 0 degrees represents the equator
`longitude`	`schema:Number`	Longitude (decimal degrees) with respect to the WGS 84 geodetic model. Ranges between −180° (west) and 180° (east) where 0 degrees represents the prime meridian
`elevation`	`schema:Number`	Elevation (m) above mean sea level with respect to the WGS 84 geodetic model
`agency`	`tatckb:Agency`	Designer, provider, or operator, of this object

each of the 22 different data types. Some of the data fields allow for more variation than others, accomplished by storing the data in the form of JSON objects. All of the data types have three base fields allowing for better organization purposes, these are `@id`, `@type`, and `@lastUpdated`. The `@id` field is assigned to the object when created and is used for a unique identifier. The `@type` field specifies the collection type the object is meant to fit into which is later used in the HTTP requests. The `@lastUpdated` field was added for a timestamp of the last time the data was changed/updated which makes finding recently manipulated data easier.

Each type has its own specified documentation for its fields, and Table 2 shows the properties, expected types, and descriptions for the `GroundStation` type of data.

Utilizing standard schema for each of the data types in the KB improves interoperability across multiple projects within the field of tradespace analysis.

Figure 5 shows the result of the request of a list of all objects in the `Agency` collection that include the regular expression "ASI." The response contains two portions: the context and the graph. The context provides guidelines for the JSON-LD schema and datatype including the specific documentation for the TAT-C KB. The graph contains the list of objects matching the request. The object contained has the fields as specified in the schema documentation for the data type `Agency`.

4.5 Containerization Configuration

Containerization refers to the ability to virtualize a development/deployment environment and isolate it from others. Containerization has recently become very large in the software industry as it allows for replicating the development environment exactly leading to less issues in deployment. Containerization allows for the separation of the back-end/database and the front-end entirely by isolating them into two separate environments.

For the containerization, Docker was the project's selected solution. The KB project includes two separate containers–one for the database and one for the web combining both the front and back-end components. The database container uses a lightweight environment specifically designed to hold a MongoDB database which was hosted on the port 27017. The web container runs a lightweight Node.js image hosted on port 80. One deployment challenge encountered here is that the web container needed to wait the MongoDB container to be fully initialized before attempting connection or it would fail generating an error. To ensure the startup sequence functioned properly, a script was written in a Dockerfile to delay the web server startup until after the database initialization.

The primary downside of using Docker on an application is that whenever changes are made to a container, it has to be rebuilt. Usually the rebuilding time is relatively quick but depending on the amount of dependencies and libraries the environment uses, it can take more time.

5 Conclusion

Architecting DMbE information systems pursues a goal of providing a flexible, reconfigurable, and updateable platform for systems engineering and design activities. This paper adopts and transitions practices from modern web-based software engineering as a reference architecture that promotes robustness while preserving changeability. Specifically, layered architectures interconnected with well-defined and constrained protocols based on web technologies such as HTTP support DMbE using principles that enabled large-scale software systems such as the Internet.

As demonstrated in the TAT-C KB application case, this paper identifies three key layers and their functionality: the database (data persistence), back-end (technical services), and front-end (user interface) components. Interfaces based on the HTTP request–response protocol provide a simple approach to access resources. Supporting data encoding standards such as JSON-LD provide enhanced semantic interoperability while preserving simple, human-readable formats. This architectural pattern can serve as the foundation for other DMbE projects that adopt alternative component implementations, interfaces, and encoding styles.

Acknowledgments This work was supported in part by NASA collaborative agreement/grant 80NSSC17K0586 titled "Knowledge Representation for Distributed Space Mission Design using TAT-C with Machine Learning" as a part of an Advanced Information Systems Technology (AIST) 2016 project.

References

Bone, M., M. Blackburn, B. Kruse, J. Dzielski, T. Hagedorn, and I. Grosse. 2018. Toward an interoperability and integration framework to enable digital thread. *Systems* 6(4). https://doi.org/10.3390/systems6040046.

Bone, M.A., M.R. Blackburn, D.H. Rhodes, D.N. Cohen, and J.A. Guerrero. 2019. Transforming systems engineering through digital engineering. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 16(4): 339–355.https://doi.org/10.1177/1548512917751873.

Cloutier, R., G. Muller, D. Verma, R. Nilchiani, E. Hole, and M. Bone. 2010. The concept of reference architectures. *Systems Engineering* 13(1): 14–27. https://doi.org/10.1002/sys.20129.

Dori, D. 2016. *Model-Based Systems Engineering with OPM and SysML*. New York: Springer.

Doyle, J.C., and M. Csete. 2011. Architecture, constraints, and behavior. *Proceedings of the National Academy of Sciences of the United States of America* 108(Supplement 3): 15624–15630. https://doi.org/10.1073/pnas.1103557108.

Fielding, R.T., and R.N. Taylor. 2002. Principled design of the modern web architecture. *ACM Transactions on Internet Technology* 62(2): 115–150. https://doi.org/10.1145/514183.514185.

Guha, R., D. Brickley, and S. Macbeth. 2016. Schema.org: Evolution of structured data on the web. *Communications of the ACM* 59(2): 44–51. https://doi.org/10.1145/2844544.

Hale, J.P., P. Simmerman, G. Kukkala, J. Guerrero, P. Kobryn, B. Puchek, M. Misconti, C. Baldwin, and M. Mulpuri. 2017. Digital model-based engineering: Expectations, prerequisites, and challenges of infusion, Technical Report NASA/TM-2017-219633, National Aeronautics and Space Administration.

Le Moigne, J., P. Dabney, O. de Weck, V. Foreman, P. Grogan, M. Holland, S. Hughes, and S. Nag. 2017. Tradespace analysis tool for designing constellations (TAT-C). In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Fort Worth. https://doi.org/10.1109/IGARSS.2017.8127168.

Office of the Assistant Secretary of Defense. 2010. Reference architecture description.

Sporny, M., D. Longley, G. Kellogg, M. Lanthaler, and N. Lindstr. 2019. JSON-LD 1.1, Standard, W3C. https://www.w3.org/TR/json-ld11/.

Digital Engineering Ecosystem for Future Nuclear Power Plants: Innovation of Ontologies, Tools, and Data Exchange

Christopher Ritter, Jeren Browning, Lee Nelson, Tammie Borders, John Bumgardner, and Mitchell Kerman

Abstract The construction of megaprojects has consistently demonstrated challenges for project managers in regard to meeting cost, schedule, and performance requirements. Megaproject construction challenges are commonplace within the nuclear industry with many active projects in the United States failing to meet cost and schedule efforts by significant margins. Currently, nuclear engineering teams operate in siloed tools and disparate teams where connections across design, procurement, and construction systems are translated manually or over brittle point-to-point integrations. The manual nature of data exchange increases the risk of silent errors in the reactor design, with each silent error cascading across the design. These cascading errors lead to uncontrollable risk during construction, resulting in significant delays and cost overruns. Additionally, due to the desire to reduce schedule and avoid escalation, construction is often begun prior to full design maturity. Digital engineering (DE) embodies a deliberate transformational approach to the manner in which systems are designed, engineered, constructed, operated, maintained, and retired. DoD defines DE as "an integrated digital approach that uses authoritative sources of system data and models as a continuum across disciplines to support lifecycle activities from concept through disposal" (U.S. Department of Defense, Digital Engineering Strategy, Washington, DC, June 2018). This paper describes the ontologies (data model), tool architectures, data exchange, and process to transform engineering teams to a new digital engineering ecosystem.

Keywords Digital engineering · Enterprise transformation · Systems engineering · Model based systems engineering · MBSE

C. Ritter (✉) · J. Browning · L. Nelson · T. Borders · J. Bumgardner · M. Kerman
Idaho National Laboratory, Idaho Falls, ID, USA
e-mail: Christopher.Ritter@inl.gov

A. M. Madni et al. (eds.), *Recent Trends and Advances in Model Based Systems Engineering*, https://doi.org/10.1007/978-3-030-82083-1_2

1 Introduction

New nuclear construction represents projects with high upfront capital costs, which have continued to increase over time. In a recent study, a team from MIT analyzed the nuclear industry's primary costs. The study compared current reactor programs over an established baseline to assess overall industry competitiveness. In the United States, V.C. Summer 2&3 in South Carolina and Vogtle 3&4 were analyzed. Neither project is predicated to meet the 2009 benchmark (https://energy.mit.edu/wp-content/uploads/2018/09/The-Future-of-Nuclear-Energy-in-a-Carbon-Constrained-World.pdf). V.C. Summer was recently canceled at a cost of over $4.9 billion to rate payers in South Carolina (https://www.chooseenergy.com/news/article/failed-v-c-summer-nuclear-project-timeline/). In Georgia, Vogtle 2&3 represent greater than $10 billion cost overrun which contributed to the bankruptcy of Westinghouse (https://www.utilitydive.com/news/southern-increases-vogtle-nuke-pricetag-by-11-billion/529682/).

Construction delays and cost overruns are not unique to the nuclear domain. The European Aeronautic Defense and Space (EADS) Airbus 380 program suffered approximately $6.5 billion in losses. Electrical wiring of airframes is a complex effort involving 530,000 meters of cables, 100,000 wires, and 40,300 connectors. During this installation, electrical teams found a critical issue – the wires were cut too short. Engineers in Germany and Spain used Dassault CATIA v4, while engineers in Britain and France had upgraded to Dassault CATIA v5. This resulted in German design teams being unable to update changes to the electrical design automatically. This interoperability issue cost Airbus 20 months of delays and a loss in program confidence (What Grounded the Airbus A380?) (Fig. 1).

Complex system issues continue to affect the aerospace, defense, and nuclear industries. Recognizing this, the Department of Defense (DoD) released a Digital Engineering Strategy (U.S. Department of Defense, Digital Engineering Strategy 2018). This strategy promotes the use of digital artifacts comprising the digital representations of systems, subsystems, and components to design and sustain national defense systems. The DoD's five strategic goals for digital engineering are to:

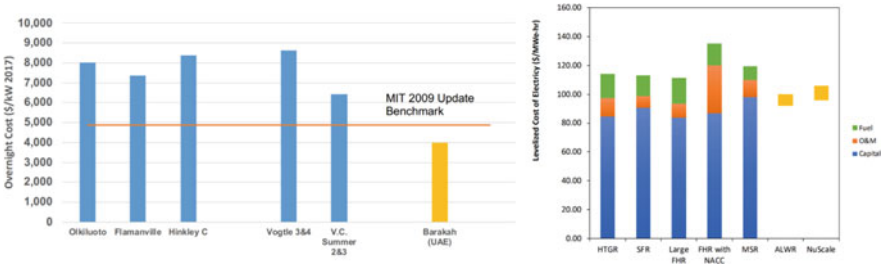


Fig. 1 (**a**) Projected LCOE for different advanced reactor concepts. (**b**) Overnight cost of recent Gen-III+ builds versus benchmark (https://energy.mit.edu/wp-content/uploads/2018/09/The-Future-of-Nuclear-Energy-in-a-Carbon-Constrained-World.pdf)