Integrated Knowledge Centric Engineering: Delivering next-generation aircraft projects at pace

Lewis Humphries
Department Of Computer Science
University of York
Edinburgh, Scotland
lewis.humphries@york.ac.uk

Gianmaria Bullegas
Leonardo UK
Edinburgh, Scotland
Gianmaria.Bullegas@leonardo.com

John Golledge
Leonardo UK
Luton, England
John.Golledge@leonardo.com

Campbell Mccausland
Leonardo UK
Edinburgh, Scotland
campbell.mccausland@leonardo.com

Donald Taylor
Leonardo UK
Edinburgh, Scotland
donald.taylor02@leonardo.com

Dimitris Kolovos
Department Of Computer Science
University of York
York, England
dimitris.kolovos@york.ac.uk

Antonio Garcia-Dominguez
Department Of Computer Science
University of York
York, England
a.garcia-dominguez@york.ac.uk

Simos Gerasimou
Department Of Computer Science
University of York
York, England
simos.gerasimou@york.ac.uk

***Abstract*—With the advent of new multi-partner, federated avionics projects for developing next-generation aircraft, there is increased pressure from the UK's Ministry of Defence (MOD) to enhance development speed, a practice known as "deliver-at-pace" [6]. This presents a challenge at both the technical and organisational levels, necessitating a move from the current heavily document-based and manual development process to a more agile and automated approach. Furthermore, the organisational culture must shift from a tool-centric mindset to a more data and knowledge-oriented one. Initiatives like openCAESAR have addressed this through the Integrated Model-Centric Engineering (IMCE) practice [8], which lays out principles for effective Model-Based Systems Engineering (MBSE). The solution proposed in this paper builds on this work, implementing these principles to create the Integrated Knowledge-Centric Engineering (IKCE) methodology, a framework for designing a modern engineering enterprise. The goal of this methodology is to solve limitations in current development processes by reducing tool reliance and increasing automation. This allows an organisation to decrease development effort, accelerate verification, and protect against vendor lock-in. Adopting IKCE will aid in achieving the MOD's "deliver-at-pace" requirements without sacrificing the rigour of the development process. By achieving this, upgrades can be implemented at a far faster pace than historically possible, moving away from rigid development processes to an inherently agile way of working. Doing this will have long-lasting benefits of the ability for future projects to change direction. This paper discusses the functions of IKCE "workspaces" and their user interactions. It uses the FireSat II satellite case study [11] to demonstrate the requirements gathering process within IKCE, showing how the framework handles various development aspects in a practical example.**

***Index Terms*—Semantic Web, Knowledge Representation, Linked Data, Continuous Integration, DevOps, Systems Engineering, Software Engineering, Model-Based Engineering, Digital Engineering**

[University of York and Leonardo UK Ltd KTP 22_23 R2, 13372, UKRI, 2023]

I. Introduction

In an effort to reduce the cost of delivering complex systems, the UK government and MOD are pushing for next-generation aircraft projects to "deliver-at-pace" [6]. This mandate requires that the time for implementing updates and changes be significantly decreased across the industry. Current development cycles involve lengthy periods for feedback and resolution, owing to the high-integrity nature of complex defence projects. To meet this demand, companies across the defence industry must examine their development processes to determine the best approach for meeting the "deliver-at-pace" requirement and overcoming the associated challenges. Section II of this paper discusses the current challenges faced in achieving this goal. Section III explains the underlying technologies used in the proposed solution. Section IV details the proposed IKCE framework. Section V presents a practical example using the solution, and Section VI concludes the paper and outlines future work.

II. Challenges

Achieving the required level of agility while maintaining the rigour essential for certification presents a significant hurdle. Traditional engineering processes were established to enforce

this rigour, often relying on extensive human review and a sequential, stage-gated progression to manage safety and risk. However, this heavy reliance on manual oversight is not only slow but also introduces its own weaknesses, including human error and inconsistent application of standards. The current implementation of these processes is often too rigid to support the required pace improvements. This approach is rooted in the waterfall life-cycle model, whereas most modern development teams now operate using an Agile model. The challenge, therefore, is not to sacrifice rigour for speed, but to implement a new approach where automation and explicit knowledge management can enhance both simultaneously. Without successfully updating these processes, the UK defence industry risks failing to deliver next-generation aircraft within the proposed time frames.

A. Technical

Data fragmentation and a lack of integration lie at the heart of the technical challenges. **Data is often siloed** at the team or project level, stored in proprietary tool formats that make it difficult to access and reuse, hindering collaboration. This leads to **vendor and user data lock-in**, where platforms retain control over user data, increasing the cost and complexity of switching tools or integrating new ones. Compounding this issue is a **lack of overarching workflow management**, which allows different projects to adopt disparate workflows, worsening the siloing effect. Consequently, development processes remain **heavily manual**, which enlarges the feedback loop for any proposed change. Finally, the presence of **multiple security levels**, each with its own network, mandates a federated approach to data management and access.

B. Organisational

Organisational inertia presents a substantial barrier to change. A fundamental **shift in mentality is required**, where teams must treat data and knowledge as their primary work products, curating them with the same care as traditional engineering artefacts. Currently, there is an excessive **focus on tools rather than the data** they produce. This is evident in practices like the **use of physical notebooks** for critical data, which is difficult to locate and impossible to search systematically, and sub-par management of documentation, which harms usability. These issues are symptoms of development processes that remain **too rigid** and rooted in the waterfall model. Furthermore, the low granularity with which data is partitioned results in **large amounts of unnecessarily classified data**, increasing the costs of both storage and development.

III. Background

To overcome the challenges outlined above, this work integrates several key technologies and methodologies. This section provides a brief overview of the core components that form the foundation of IKCE.

A. Semantic Web Technologies (RDF & OWL)

The Resource Description Framework (RDF) [12] is a foundational data model for representing information as triple statements (subject-predicate-object) in a directed graph. This structure allows data to be stored in a flexible format within an RDF store and queried using the SPARQL language [2]. Building upon RDF, the Web Ontology Language (OWL) [1] is used to create ontologies—formal specifications of a domain's classes, properties, and the relationships between them. OWL enhances RDF by enabling inferencing, which allows for the automatic deduction of new facts from existing data. These technologies provide the basis for the vendor-independent data interchange format.

B. Ontological Modelling Language (OML) and openCAESAR

The Ontological Modelling Language (OML) [8] is a formalisation of a subset of OWL, specifically designed to bridge open and closed-world data models in a systems engineering context. OML was developed by the openCAESAR project [8], [9], a set of tools and transformations built around OML and OWL that promotes a practice for MBSE known as Integrated Model Centric Engineering (IMCE). IMCE outlines the key principles and characteristics of a successful MBSE process, which heavily influenced the design of IKCE.

C. Model-Driven Engineering (MDE) with Epsilon

Epsilon [13] is a family of languages and tools for Model-Driven Engineering (MDE) that facilitates the automation of tasks such as model validation, transformation, and generation. It integrates with common modelling technologies like the Eclipse Modelling Framework (EMF) and Unified Modelling Language (UML), providing the technical means to create the tool adapters required by IKCE.

D. Containerised Development Environments

A containerised development environment uses technology like Docker to encapsulate the tools, libraries, and configurations needed for development. This ensures consistency across different machines, simplifies the onboarding of new engineers, and prevents configuration conflicts between projects. For this Coder [4] is used, which is a platform for managing these environments from customizable templates, to provide engineers with consistent and project-specific toolsets.

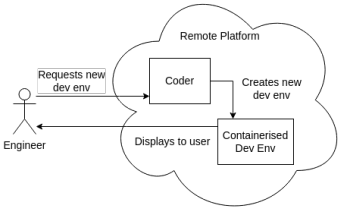

Fig. 1. Process for using Coder with Containerised Development Environments

Figure 1 shows a simplified example of an engineer using coder to request a new development environment.

IV. Proposal: IKCE

IKCE represents a new approach to systems engineering that leverages formal knowledge representations (ontologies) and standardised artefact exchange. It fosters collaboration through a publish-subscribe network, enabling access to the right information at the right time while automating version control and change management. This methodology moves away from rigid, human-dependent processes toward a machine-interpretable, dynamic system. The IKCE framework, created based on the principles of IMCE, provides a unified approach to managing and integrating system knowledge by adhering to the following principles:

- **Authority/Domain Driven Federation**: System data is federated and belongs to specific domains based on system and work breakdown structures. Each discipline continues to use familiar tools to produce domain-specific models and documents. These information artefacts are managed using version control and change management within their native environments. An enterprise-level data and workflow orchestration service connects to these disparate data sources and provides a uniform interface over them. This service is realised through a combination of CI/CD pipelines and a message bus that listens for changes in source repositories and triggers the appropriate data transformation and propagation workflows.
- **Knowledge Independent of Representation**: System knowledge resides in information artefacts that are enriched with metadata connecting them to the broader system knowledge graph. This metadata is captured in a standard, vendor-independent format (OML), which is integrated into a Knowledge Graph that serves as the authoritative source of truth for reference throughout the development lifecycle.
- **Logical vs. Temporal Precedence**: IKCE enables an agile approach to large-scale systems development based on flexible workflows driven by the logical dependencies between system functions, rather than a rigid temporal relationship between work packages. For example, a traditional waterfall model dictates that all systems requirements must be completed (a temporal constraint) before software design can begin. In contrast, IKCE allows the software team to start designing a user interface as soon as the relevant UI requirements are published (a logical dependency), even if requirements for other subsystems are still being defined.
- **Self-serve Data Availability**: Data is discoverable and accessible throughout the organisation via a secure self-service portal, reducing the time taken to locate required information.
- **Data Governance at the Source**: Data governance is applied where the data is created and produced to ensure the correct implementation of security and IP concerns. This makes data quality and security a shared responsibility, rather than the task of a centralised team with limited knowledge of the data being governed.

A. How It Works

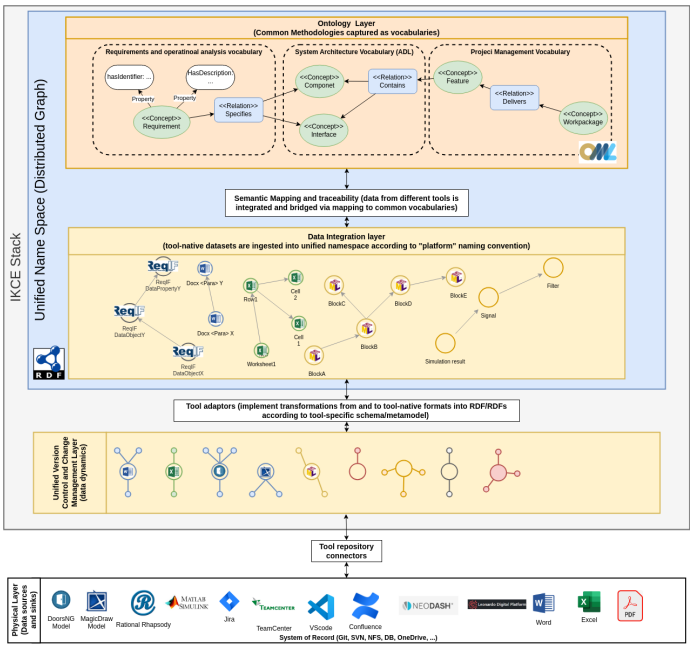

Fig. 2. Overview of the IKCE Information Architecture

1) The IKCE Stack: The IKCE information architecture (Figure 2) is designed to create a unified knowledge graph from diverse and disconnected data sources. The core of the architecture is the IKCE Stack, a distributed graph that functions as a unified namespace. It is composed of five key layers :

Physical Layer: This layer consists of the various engineering tools (e.g., DOORS, Cameo) and data sources (e.g., Git repositories) used in projects.

Tool Adapters Layer: This layer is responsible for converting data from proprietary tool formats into a standardised RDF-based format and managing version control for the incoming data.

Data Integration Layer: The standardised data is ingested into this layer, where individual data elements (requirements, design components, etc.) are organised according to a common naming convention.

Semantic Mapping and Traceability Layer: This layer creates meaningful connections between different data elements by mapping the raw data to a set of common, predefined vocabularies.

Ontology Layer: This top layer provides the formal, shared vocabularies (ontologies) that define the core concepts and relationships across different domains, enabling semantic mapping. Figure 2 shows three example vocabularies:

- Requirements and operational analysis (e.g. Concept: Requirement)

- System Architecture (e.g. Concept: Component, Relation: Contains)
- Project Management (e.g. Concept: Feature, Relation: Delivers)

In essence, the IKCE Stack transforms data from various tools into a standard format and then uses a common set of ontologies to build a connected, traceable, and semantically rich knowledge graph. While Description Logic (DL) reasoning within this layer can validate classifications and relationships, it does not replace the specialised analysis (e.g., physics-based simulations, geometric analysis) performed by tools in the physical layer. Instead, IKCE serves as a connective framework to integrate the inputs and outputs of these tools into a coherent system model.

2) Data Transformations: To ensure that IKCE is tool agnostic, a process for transforming data from tool-based formats into the common exchange format has been developed. For this process the common exchange format chosen is OML, making use of a mix of Eclipse Epsilon transformations [13] and the OML adapter library [8] for connecting to the various tools used.

3) Foundational Vocabularies: To aid in the processes of mapping between data developed in different projects, a selection of ontologies has been created at the enterprise level which are known as foundational vocabularies. These ontologies consist of foundational concepts at a high level of abstraction which are designed to be built on and specialised for the specific domains, as shown in Figure 3. This will allow for taking data from a domain that uses a foundational vocabulary and using it within different domains that also have specialisations of that same vocabulary. By doing this the amount of mapping vocabularies that are required will be reduced since using a common vocabulary as a base gives some of the mappings between different data for free.

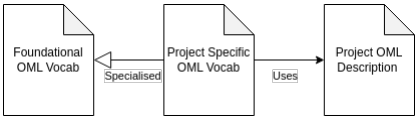

Fig. 3. Diagram Showing How Foundational Vocabularies Are Used

4) Mapping Data Models: For cases where having a common vocabulary is not enough to link data from different sources and domains together, mapping vocabularies can be created. Mapping vocabularies are ontologies that contain all the various pieces of logic that link the types of nodes from different ontologies together. Figure 4 shows an example of using mapping vocabularies to link three different OML vocabularies representing the data models of ReqIF, a project-specific model, and a foundational model. This allows for data stored in an OML description based on the ReqIF vocabulary, project-specific requirements vocabulary, and foundational vocabulary to be treated the same using inferencing due to the links created by the mapping vocabularies.

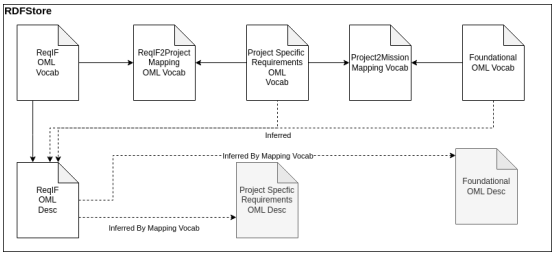

Fig. 4. Diagram Showing How Different Data Models Are Mapped

5) Workspaces: Once the IKCE Stack has been put in place, the next key component of IKCE is workspaces, which is a logical separation of tools and data within the wider enterprise. Figure 5 is an overview of an example workspace, showing how users use the development tools, which then send the data to a transformation stage, converting it into the common interchange format and sending the transformed data to the knowledge graph ready to be published. These are typically created to perform a specific task within a larger project which are used by either individuals or entire teams.

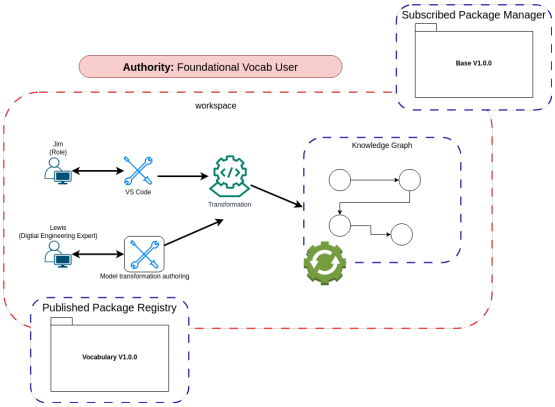

Fig. 5. Diagram Showing Overview of a Workspace

Workspaces contain multiple key types of data and processes which are used by workspaces to function, which are shown in Figure 6 and discussed in more detail below:

- Data Source: Data sources are the various places in which data used within a workspace comes from, such as repositories and package registries.
- Resource: A resource represents a piece of data used by a workspace, such as a git project.
- Package Registry: A package registry is a type of repository that stores packages published by other workspaces, which a new workspace can then subscribe to.
- Package: A package is a piece of data that has been published by a workspace.
- Dependency: A dependency represents a link to a package that a workspace has subscribed to.
- Workflow: A workflow represents a process used within a workspace.

- Pipeline: A pipeline is a specific type of workflow within a workspace that represents a CI/CD pipeline.
- View Point: A viewpoint represents a user interface for data within a workspace.
- Development Environment Template: A development environment template is the template that is used to instantiate a containerised development environment for users.
- Development Environment: A development environment is the set of tools that are used by users to complete whatever tasks the workspace was created for.
- Tool: A tool is a part of the development environment that helps the user in completing tasks.

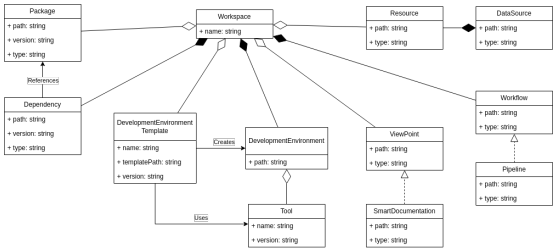

Fig. 6. Diagram Showing the Data Model for a Workspace

Example Workspace

- Master Resource: Ephemeral OML project containing all data within a workspace.
- Development Resource: Contains the data currently being developed within the workspace.
- Development Environment Resource: Contains the template for generating containerised development environments
- Smart Documentation Resource: More user-friendly visualisation of data contained within the master resource
- CI/CD Pipeline: Various workflows for doing tasks such as publishing packages to package registry
- Package Registry: Type of repository that contains published packages to be used by other workspaces.

6) Templates: To allow for quickly creating new workspaces designed for common tasks and workflows within the development processes, a templating mechanism has been developed. This is used to provide a skeleton for common types of workspaces that can be tailored to fit the needs of specific tasks. This tailoring process allows for flexibility within IKCE, reducing the risk of teams attempting to bypass the processes put in place since the templates can be modified to mitigate issues with the standard approach to certain task. These templates can then be discovered through the discovery portal when creating a new workspace.

7) Pub/Sub Collaboration: Once a collection of IKCE workspaces have been setup to handle various tasks within a project, the next question is how do they collaborate with each other. The approach taken for collaboration is to use a publish and subscribe approach, in where a workspace will publish work that it has completed to its own package registry, which other workspaces can then subscribe to. Figure 7 shows an example of this process between two IKCE workspaces, where a developer of the base package makes a change and publishes an updated version to the published package registry, which is then subscribed to by the other workspace. This creates a link showing how different workspaces depend on each other for various components, which acts as a logical precedence between workspaces, showing the order in which workspaces need to be created. This approach changes the overall paradigm of the development processes to a more inherently agile and flexible publish and subscribe process, compared to the rigid assembly line process.

8) Discovery Portal: To aid teams in reusing existing data within the wider enterprise, a discovery portal has been developed to search for data that has been published by other workspaces within the organisation that can then be subscribed to within the workspace. This will allow for searching across the entire organisation's collection of available data, which can then be subscribed to and reused by new workspaces. The discovery portal will also allow for locating templates for creating new workspaces based on common processes used within the organisation. This reuse will allow for both reducing the amount of development time and cost for new projects, while also reducing the siloing across projects, due to the data from one project being usable in another.

B. How Users Interact With IKCE

Another concern is the approach taken by the various different types of users within the organisation when using IKCE. In most cases, engineers will not need to change their workflow significantly with IKCE, since one of the goals of IKCE is to allow for using the current tools in place. This will work by doing the various different data transformations required for linking tools together within the background, being almost invisible to users in their day to day workflow.

1) Workspace Management UI: For the situations where users require more involved interaction with key IKCE concepts, a UI has been developed that allows for working with the various aspects of IKCE. Figure 8 shows how the UI is used to interact with IKCE, which consists of two main services, which will handle the admin and user level functionality for workspaces respectively.

IKCE Workspace Admin Service: The IKCE Workspace Admin Service handles admin-level functions for workspaces, such as the creation and modification of workspaces.

- Create new workspaces
- Modify existing workspaces
- Add/Remove Users from workspaces

IKCE Workspace User Service: The IKCE Workspace User Service handles the user-level functions of workspaces, such as viewing the data within a workspace and creating new development environments within a workspace.

- Create new development environments
- Access existing development environments
- Approve pull requests
- View data within a workspace
- See available workspaces

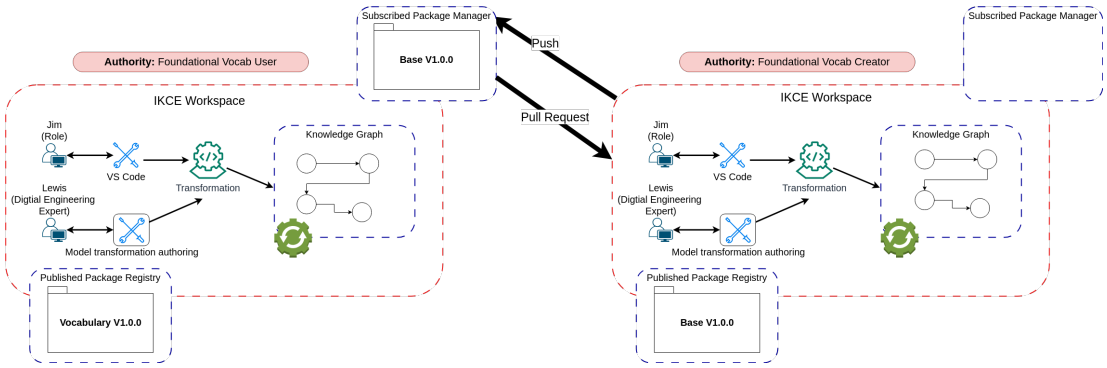

Fig. 7. Diagram Showing How Workspaces Publish/Subscribe to Packages

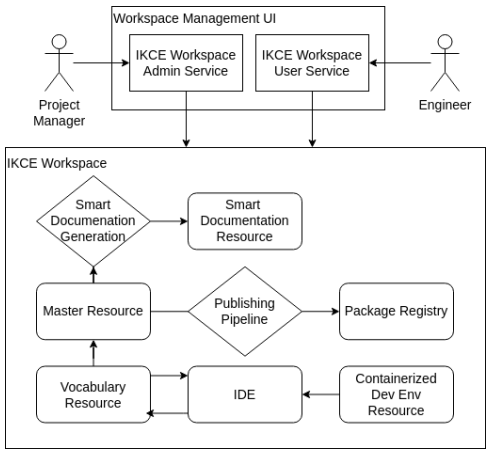

Fig. 8. Diagram of How the User Interacts with IKCE

2) Workflows: To show how users would interact with IKCE using the UI, the workflows for workspace creation (Figure 9), containerised development environment setup (Figure 10), and modifying data within a workspace (Figure 11) are shown below.

Workspace Creation: For the creation of a new workspace within IKCE, first the Project Manager who will manage the Workspace needs to go to the IKCE Workspace Admin Service and request a new workspace.

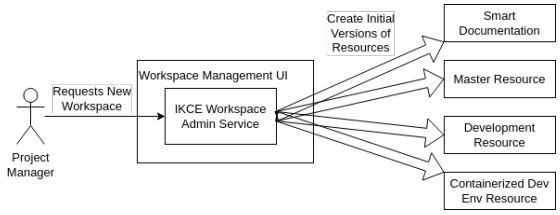

Fig. 9. Workflow for Creating IKCE Workspace

This will then show a form with the following fields about the new workspace that the user will need to fill in:

- Workspace Name - The namespace data created inside the workspace will use.
- Workspace Template - The template that the workspace will be built from.
- Workspace Dependencies - What packages the newly created workspace will depend on.
- Workspace Members - Who has access to new workspace.

Once the form has been filled in and submitted, the Workspace Admin Service will then send this information to the backend, which will create all the required resources and pipelines for the workspace, which is determined by the selected template.

Development Environment Creation: When a Developer within a workspace needs a new Containerised Development Environment, first they will go to the IKCE Workspace User Service and request a new Development Environment. This will send a request to the Containerised Development Environment Manager to setup a new Development Environment based on the template for the chosen workspace. The Containerised Dev Env Manager will then create the Development Environment and send a reference back to the UI, which will then give the Developer access to the newly created Development Environment.

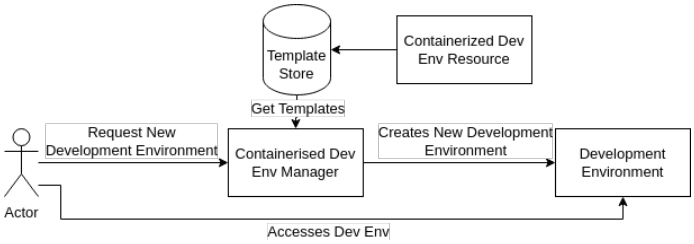

Fig. 10. Workflow for Creating IKCE Development Environments

Making Changes To Workspace Data: Once the Workspace and Development Environment has been created, the next step would be to make changes to the data stored within the new Workspace. To do this first the Developer will request access to their Development Environment using the IKCE Workspace User Service, which will send that request to the Containerised Development Environment Manager. This will go and retrieve the specified Development Environment and be sent back to the UI where the Developer will be given access, ready to make their required changes. Once the Developer has their Development Environment, they will

pull down the Git branch required for their task from the Development Resource. They will then use whichever Development Tool is required for the specific task being done to modify the data within the Workspace. Once this is done, the changes are committed onto the branch and pushed back to the Development Resource, where a Review will receive a pull request and either approve or deny the changes. If the changes are approved and pushed to the main branch, a pipeline will automatically start that will take the changes to the Development Resource and add them to the Master Resource. This will trigger multiple other pipelines that will propagate the changes across the workspace, with some examples being updating the Smart Documentation Environment, or publishing a new version of the data to the Workspace's Package Registry.

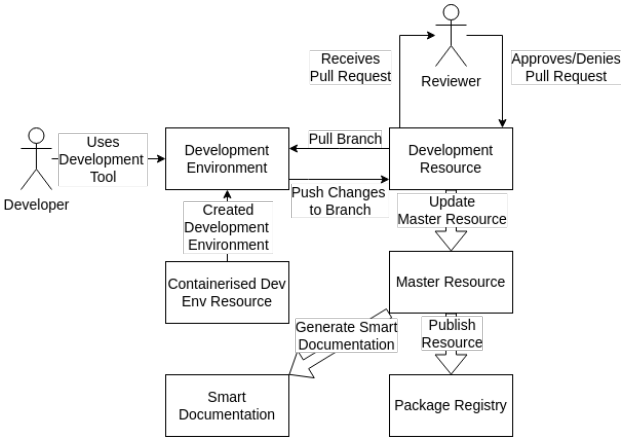

Fig. 11. Workflow for Making Changes in an IKCE Workspace

V. FireSat Example

A. The Problem

This example, based on the FireSat II satellite [11], focuses on the requirements gathering process and its transfer from the systems engineering to the software engineering domain.

B. Primary Goals

- To reduce the impact of wildfires in the US and Canada.
- To reliably and efficiently operate a space-based asset that provides critical fire data.

C. Stakeholders

Forest Service (End User):

- Concerns: Primarily concerned with the "loss of life and property due to forest fires."
- Needs: Ability "to put out a fire in a timely manner to prevent the loss of life and property." This implies a need for rapid detection and accurate location data.

Fire Department (End User):

- Concerns: The operational response to a fire.
- Needs: Receive "forest fire data in real time" to effectively dispatch resources and manage firefighting efforts on the ground.

Sponsor (Customer):

- Concerns: Overall value and viability of the project.
- Needs: "establishing a feasible design to satisfy the mission requirements within cost and schedule constraints". They are the source of the funding and expect a return on investment.

Development Contractor (Supplier):

- Concerns: Successfully building and delivering the system.
- Needs: "satisfy the mission requirements" while operating "within cost and schedule". Clear, unambiguous technical specifications required to build against.

Operator (User):

- Concerns: Day-to-day operation of the spacecraft.
- Needs: "operate the spacecraft to achieve the mission objectives". This includes the ability to "monitor and maintain the health and safety of FireSat II" and requires a well-defined command and telemetry interface.

D. Scenario

The scenario will focus on the processes around the gathering of requirements and their transfer from the systems engineering to software engineering domains, which consists of the three key stages as shown below:

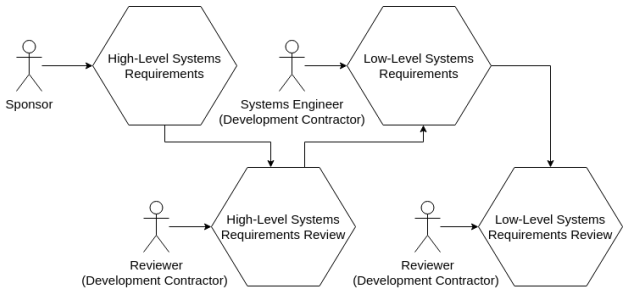

Fig. 12. First Stage of the System to Software Requirements Process

Figure 12 shows the first stage, where the high-level systems engineering requirements are received by the customer, which are then reviewed and broken down into low-level systems engineering requirements.

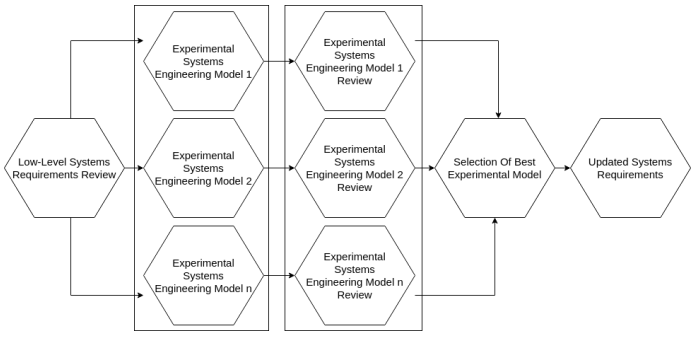

Fig. 13. Second Stage of the System to Software Requirements Process

Figure 13 shows the next stage which happens once the low-level systems requirements have been created and reviewer,

wherein multiple different experimental systems engineering models are created and reviewed to see which model best suits the requirements given. Once a model is chosen, it's requirements are then compared against the original systems requirements, which are then updated to reflect the chosen model.

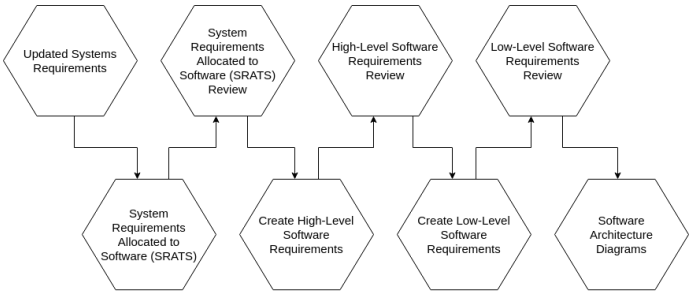

Fig. 14. Third Stage of the System to Software Requirements Process

Once the final systems requirements have been created, the final stage (Figure 14) involves going through the requirements and extracting everything that is applicable to the domain of software engineering, which are known System Requirements Allocated to Software (SRATS). The high-level software requirements will then be created based off the identified SRATS, which will be reviewed and used to create the low-level software requirements, similar to how the low-level systems requirements are created. Once the low-level software requirements have been reviewed, they are used in the creation of the diagrams showing the architecture of the software, using formats such as UML or C4.

E. Implementation

The implementation follows the third stage of the requirements process (Figure 14) shown above, implementing it within IKCE. For this example, the tools used are IBM DOORS for requirements management, Cameo for system modelling, XText for a custom review DSL, and the C4 model for software architecture diagrams.

1) Tools: IKCE is designed to support any tool with an adapter for transferring data from the tool to the common interchange format, but for this example the following tools have been selected:

DOORS: IBM DOORS [3] is a tool used for capturing and managing the requirements used in the development of projects.

Cameo: Cameo [7] is a system modelling tool that is typically used by systems engineers to create models of the product being developed for a project, but it also has some capabilities of storing and editing requirements.

XText: XText [10] is a software framework that is designed for developing domain-specific language, which are programming languages that have been created for a specific purpose. In this scenario, XText has been used to create a DSL that allows for reviewing requirements.

C4: For designing the software architecture, the C4 model [5] has been used, which is a framework for designing architecture diagrams for software systems.

2) Workflow: To represent this, the process will be split into workspaces based on the following tasks:

Updated Systems Requirements: In this first step, a workspace (Figure 15) is created with the goal of taking the chosen experimental systems model and creating the final set of systems requirements that will be used for allocating requirements to software.

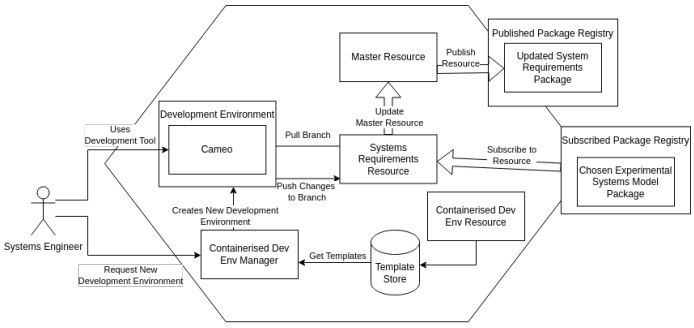

Fig. 15. Overview of Workspace Used For Updating Systems Requirements

This works by first creating a new workspace that subscribes to the Chosen Experimental Systems Model package. Once the workspace has been created, the systems engineer will then need to request a new development environment for that workspace, which in this instance will contain an instance of Cameo.

#	Id	Name	Text
1	34	Mission Requirements-SMAD Table 3-4	
2	34.1	Performance	
3	34.1.1	Weather	Work through light clouds
4	34.1.2	Resolution	50 meter resolution
5	34.1.3	Geo-location Accuracy	1 km geolocation accuracy
6	34.10	Data Distribution	Up to 500 fire-monitoring offices + 2,000 rangers worldwide (max of 100 simultaneous users)
7	34.11	Data Content, Form, and Format	Location amd extent in lat/long for local plotting. avg temp for each 40 m2
8	34.12	User Equipment	10 X 20 cm display with zoom and touch controls, built-in GPS quality map
9	34.13	Cost	Non-recurring<$10M Recurring<$3M/year
10	34.14	Schedule	Operational within 3 years
11	34.15	Risk	Probability of success>90%
12	34.16	Regulations	Orbital debris, civil program regulations
13	34.17	Political	Responsive to public demand for action
14	34.18	Environment	Natural
15	34.19	Interfaces	Interoperable through NOAA ground stations
16	34.2	Coverage	Coverage of specified forest areas within the US at least twice daily.

Fig. 16. Updated Systems Requirements within Cameo

Once the user has the development environment, they will then pull down the required branch from the System Requirements resource into Cameo (Figure 16), which will transform the data from OML into Cameo's data model using an adapter.

Once that is done, the user will modify the systems requirements based on the chosen model and push their changes back to the System Requirements resource, transforming the data back into OML where it is ready to be reviewed and sent to the master resource (Figure 17).

System Requirements Allocated To Software (SRATS): Now that the system requirements have been updated, the next step is to extract the requirements that are applicable to software, known as SRATS.

```
instance MissionRequirements : mission:Requirement [
    base:hasIdentifier "FRC-REQ.1"
    base:hasCanonicalName "Firesat Mission requirements set"
    base:hasDescription ""
    analysis:appliesDuring sc:always
]

instance PerformanceReq : mission:Requirement [
    base:hasIdentifier "FRC-REQ.1.1"
    base:hasCanonicalName "Performance"
    base:hasDescription ""
    analysis:appliesDuring sc:always
    analysis:requires aConstraint
    base:isAggregatedIn MissionRequirements
]

instance WeatherReq : mission:Requirement [
    base:hasIdentifier "FRC-REQ.1.1.1"
    base:hasCanonicalName "weather"
    base:hasDescription "Work through light clouds"
    analysis:appliesDuring sc:always
    analysis:requires aConstraint
    base:isAggregatedIn PerformanceReq
]
```

Fig. 17. Updated Systems Requirements as an OML Description

This will begin by creating a new workspace (Figure 18) that subscribes to the updated systems requirements package, where a new development environment containing a DOORS instance will be created.

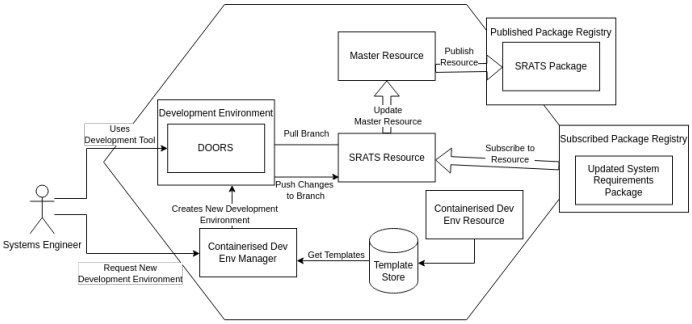

Fig. 18. Overview of Workspace Used For Extracting SRATS

Once in this new DOORS instance, the required branch in the SRATS resource will be pulled down and transformed from OML to the DOORS data model (Figure 19). The user will then extract the SRATS and push the changes back to the SRATS resource when completed, transforming the data back to OML (Figure 20).

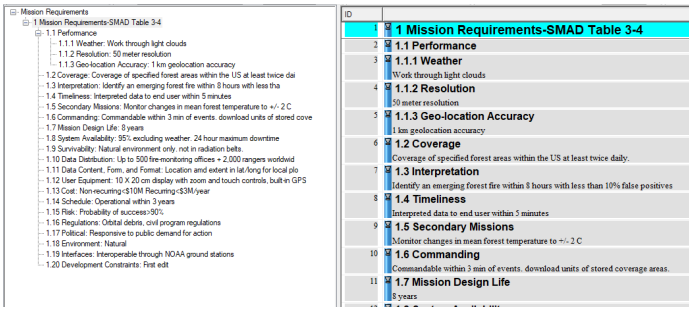

Fig. 19. Updated Systems Requirements in DOORS

The SRATS will then be reviewed and if accepted pushed to the master resource and published to the package registry in the same way as the updated systems requirements were.

```
instance SensorCommandReceivedReq : mission:Requirement [
    base:hasIdentifier "SRAT-REQ.101"
    base:hasCanonicalName "When Sensor On Command is received"
    base:hasDescription "When Sensor On Command is received"
    analysis:appliesDuring sc:always
    analysis:requires aConstraint
]

instance SensorHardwareOnReq : mission:Requirement [
    base:hasIdentifier "SRAT-REQ.101.1"
    base:hasCanonicalName "Turn on the Sensor Hardware"
    base:hasDescription "Turn on the Sensor Hardware"
    analysis:appliesDuring sc:always
    analysis:requires aConstraint
    base:isAggregatedIn SensorCommandReceivedReq
]
```

Fig. 20. SRATS as an OML Description

System Requirements Allocated To Software (SRATS) Review: The next workspace to be created is for the review of the selected SRATS, which will make use of a custom domain-specific language (DSL) that was created with XText, using adapters in the same way as the previous workspaces to convert between OML and the review DSL.

```
review SRATSREview
    Date:"1st July 2025"
    Time:"13:05"
    Duration:"60"
    Artefact(s):SensorCommandRecievedReq, SensorHardwareOnReq
    Author(s):SystemsEngineer1
    Moderator:Reviewer1
    Recorder:Reviewer2
    Reader:SystemsEngineer2
    Upstream:SystemsEngineer2
    Downstream:SoftwareEngineer1
    Reinspect:TBD
    ---ID---+-Severity-+--Type---+--Raised--+---On-------------------+-Detail-----------------------------
    - Observation1 COS SUGGEST Reviewer1 General "Create a set of required conditions for enabling sensor";

    -Resolves-+-Resolution-+-Accept-+-Accepted-+-Comment-------------------------------------------------------
    Observation1 ACCEPT YES SystemsEngineer1 "Conditions for enabling sensor to be added to this requirement"
end
```

Fig. 21. Review for SRATS within the Review DSL

When the data is in the editor for the review DSL, the reviewer will analyse the SRATS for any potential concerns and issues, recording any observations in a report using the format shown in Figure 21.

Once the review is complete, the data is sent from the SRATS review resource to the master resource and published, where it can be used to create high-level software requirements if accepted, or used to help with the process of fixing any identified issues in the SRATS if not.

Create High-Level Software Requirements: This next workspace subscribes to the reviewed SRATS and puts them into DOORS, where the user can analyse them and create a new set of high-level requirements that are purely software-focussed. These are then sent to the High-Level Software Requirements resource and published using the same process as above.

Review High-Level Software Requirements: The high-level requirements are then reviewed the same way as the SRATS were, using a workspace containing DOORS where the high-level requirements package is subscribed to.

Create Low-Level Software Requirements: Now the high level software requirements have been reviewed, this process is repeated for the low-level software requirements. This consists of a workspace that subscribes to the reviewed high-level software requirements and uses DOORS to create more focused lower level software requirements. The data is then

pushed back to the low level software requirements resource and published the same way as before.

Review Low-Level Software Requirements: These low-level software requirements are then reviewed by a workspace in the same manner as the SRATS and high-level software requirement were.

Software Architecture Diagrams: A final workspace is created that subscribes to the reviewed low-level requirements that is used to create a software architecture diagram using the C4 model, as shown in Figure 22.

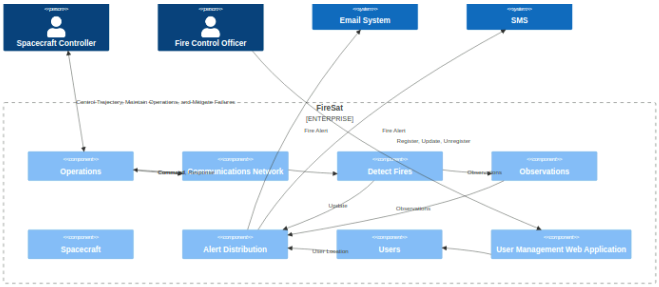

Fig. 22. C4 Container Diagram for FireSat

Once created, this model will be pushed to the software architecture resource and sent to the master resource and published in the same way as the previous workspaces, ready to be used in the next stage of the software engineering domain. This completes the process of going from the systems engineering domain to software engineering domain.

VI. Conclusion and Future Work

This paper presents the Integrated Knowledge-Centric Engineering (IKCE) approach, designed to address the challenges faced by the defence industry in meeting the MOD's "deliver-at-pace" mandate. By moving from rigid, manual development processes to a more flexible and automated framework, IKCE reduces the size and number of feedback loops within large-scale avionics projects. The methodology provides a path to increasing development velocity without compromising the rigour required for high-integrity systems.

Future work will focus on maturing these concepts and beginning their implementation within the organisation. A key activity will be to increase the number of supported tool adapters to enable more engineering domains to benefit from the IKCE framework. The creation and maintenance of domain ontologies are also recognised as a significant undertaking. Therefore, there are plans to investigate how AI, particularly Large Language Models (LLMs), could be leveraged to assist in generating and refining ontologies from existing engineering documents and unstructured data. Furthermore, another point that needs addressing are the challenges of scalability. While the federated workspace architecture is designed to be more scalable than a single monolithic knowledge graph, studies to analyse performance and adoption hurdles in a full-scale industrial deployment will need to be conducted.

References

[1] Owl 2 web ontology language document overview. W3C Recommendation, 10 2009. Accessed: 2025-07-07.

[2] SPARQL 1.1 Query Language. Technical report, W3C, 2013. Accessed: 2025-07-07.

[3] Ibm engineering requirements management doors. Software, 2024. Accessed: 2025-07-07.

[4] Coder : Cloud development environment: Remote & self hosted, 2025. Accessed: 2025-07-07.

[5] Simon Brown. The c4 model for visualising software architecture. *Context, Containers, Components, and Code. URl: https://c4model.com/.(accessed: 07.07.2025)*, 2018.

[6] Defence Comittee. Report favc0018: Written evidence submitted by the ministry of defence (pdf), November 2023.

[7] Dassault Systemes. Cameo Systems Modeller, 2024. Accessed: 2025-07-07.

[8] Maged Elaasar, Nicolas Rouquette, Steve Jenkins, and Sebastien Gerard. The case for integrated model centric engineering. *Proceedings of the 10th model-based enterprise summit (MBE 2019). National Institute of Standards and Technology, Gaithersburg, MD*, pages 9–16, 2019.

[9] Maged Elaasar, Nicolas Rouquette, David Wagner, Bentley James Oakes, Abdelwahab Hamou-Lhadj, and Mohammad Hamdaqa. opencaesar: Balancing agility and rigor in model-based systems engineering. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 221–230. IEEE, 2023.

[10] Moritz Eysholdt and Heiko Behrens. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309, 2010.

[11] Sanford Friedenthal and Christopher Oster. *Architecting Spacecraft with SysML: A Model-Based Systems Engineering Approach*. CreateSpace Independent Publishing Platform, 2017.

[12] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation, 2004. Accessed: 2025-07-07.

[13] Dimitrios Kolovos, Louis Rose, Richard Paige, and A Garcia-Dominguez. *The Epsilon Book*. Eclipse, 2010. Accessed: 2025-07-07.