Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

HyperGraphOS: A Meta Operating System for Science and Engineering

Antonello Ceravola, Frank Joublin, Ahmed Sadik, Bram Bolder, Juha-Pekka Tolvanen

2024

Preprint:

This is an accepted article published in MODELSWARD 2025. The final authenticated version is available online at: https://doi.org/[DOI not available]

HyperGraphOS: A Meta Operating System for Science and Engineering

Antonello Ceravola[1][a], Frank Joublin[1][b], Ahmed R. Sadik[1][c], Bram Bolder[1][d] and Juha-Pekka Tolvanen[2][e]

[1]*Honda Research Institute Europe, Offenbach, Germany*

[2]*MetaCase, Jyväskylä, Finland*

{*antonello.ceravola, frank.joublin, ahmed.sadik, bram.bolder*}*@honda-ri.de, jpt@metacase.com*

Keywords: Operating Systems, Agile Model-Based System Engineering, Graph-Based Modelling, Domain Specific Languages, Artificial Intelligence

Abstract: This paper presents HyperGraphOS, an innovative Operating System (OS) designed for the scientific and engineering domains. It combines model-based engineering, graph modeling, data containers, and computational tools, offering users a dynamic workspace for creating and managing complex models represented as customizable graphs. Using a web-based architecture, HyperGraphOS requires only a modern browser to organize knowledge, documents, and content into interconnected models. Domain-Specific Languages (DSLs) drive workspace navigation, code generation, AI integration, and process organization. The platform's models function as both visual drawings and data structures, enabling dynamic modifications and inspection, both interactively and programmatically. HyperGraphOS was evaluated across various domains, including virtual avatars, robotic task planning using Large Language Models (LLMs), and meta-modeling for feature-based code development. Results show significant improvements in flexibility, data management, computation, and document handling. By bridging traditional OS functionality with innovative UX design to fulfill the needs of modern applications, HyperGraphOS delivers enhanced productivity and efficiency. Its graph-based model representation and integration with DSLs create a highly flexible, user-friendly environment, making it ideal for a wide range of scientific and engineering contexts.

1 INTRODUCTION

Operating Systems (OSs) have evolved significantly since the 1950s, when they were first developed for general-purpose computers such as IBM's 701 and 709, as illustrated in Figure 1. Initially, these systems required manual intervention for executing programs and lacked automation. The introduction of *batch processing* in the 1950s, exemplified by IBM systems, allowed sequences of jobs to be processed without human input. *Time-sharing* systems soon followed, enabling multiple users to interact with the same computer simultaneously, as seen in MIT's CTSS and IBM's System/360. At this same time, Teletypewriters (TTY) and the concept of *file* were introduced, followed a few years later by the concept of hierarchical *folders* in Multics. At the end of the 1960s, UNIX, developed at Bell Labs, popularized these abstractions and the concept of *console* which since then form the foundation of modern OSs.

The rise of personal computers in the 1980s, along with OSs like MS-DOS and Windows, brought computing to individual users, and popularized the WIMP (*Windows*, *Icons*, *Menus*, *Pointer*) paradigm developed and experimented in the 1970s. *Graphical User Interfaces* (GUIs) further simplified interactions by introducing these visual elements. In the 2000s, OSs expanded to support mobile devices and cloud-based platforms like iOS, Android, and Chrome OS, while still relying on core UNIX-based abstractions. Despite advances in security, processing power, and interface design, general-purpose OSs remain limited in addressing the specific needs of specific users, which demand more advanced data manipulation and model integration tools.

This research introduces HyperGraphOS, a web-based OS designed specifically for scientific and engineering contexts. The OS is publicly available on GitHub (HRI-EU, 2024a), enabling researchers

[a] https://orcid.org/0000-0002-1075-459X
[b] https://orcid.org/00000-0002-4421-1737
[c] https://orcid.org/0000-0001-8291-2211
[d] https://orcid.org/0009-0002-5595-2466
[e] https://orcid.org/0000-0002-6409-5972

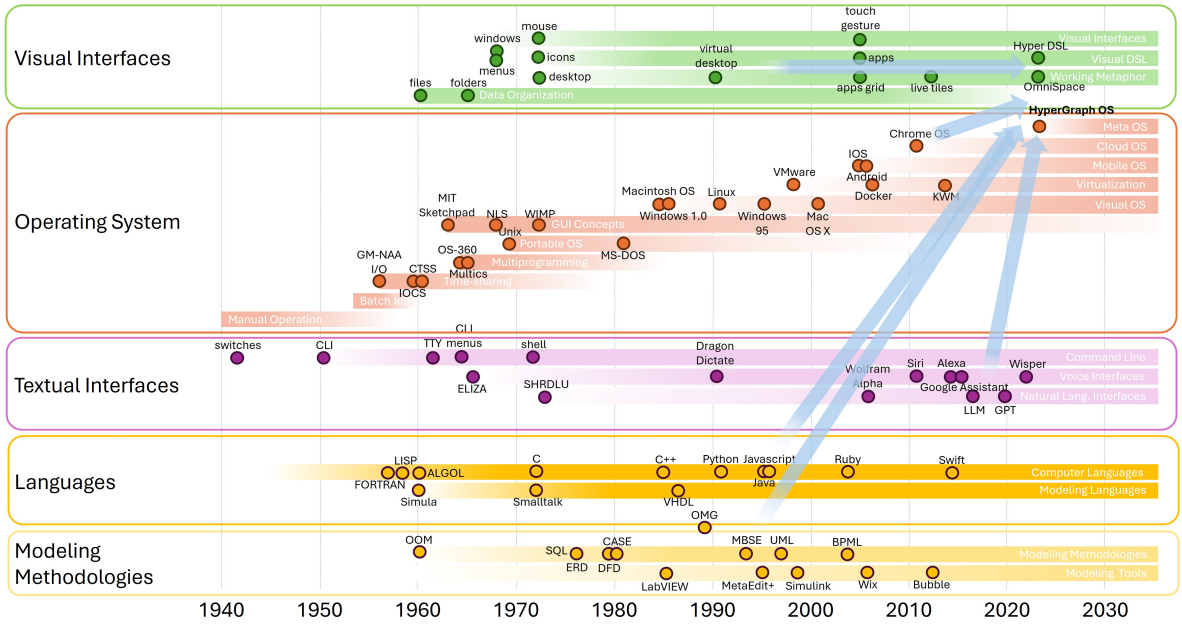

Figure 1: Historical Influences leading to the development of HypergraphOS

and developers to contribute and extend its capabilities. HyperGraphOS leverages DSLs and graph-based models to provide a flexible and efficient environment for managing complex data and models, addressing the limitations of current general-purpose OSs. Integrating advanced technologies such as Artificial Intelligence (AI) and Large Language Models (LLMs), HyperGraphOS allows dynamic interaction with data, both interactively and programmatically. The system's unique design offers infinitely connected WorkSpaces (i.e., *workspace*)(HRI-EU, 2024b), customizeable semantics, and visual data representations, addressing the limitations of traditional OSs in specialized domains. HyperGraphOS is built on a minimal kernel with fractal design principles (Lorenz, 2002)(Blair et al., 2009)(Ediz, 2009), ensuring extensibility and adaptability.

This paper is not intended to cover all the characteristics of HyperGraphOS but rather aims to give a first impression of it. For a more extensive and comprehensive analysis, several other articles are in preparation, each focusing on a different perspective. The rest of this paper is organized as follows: Section 2 outlines the evolution of OSs, Section 3 details the main concept behind HyperGraphOS, and Section 4 explores its architecture and core features. Section 5 presents case studies demonstrating its practical application, and the paper concludes by comparing HyperGraphOS to existing systems, highlighting its unique contributions and potential future developments.

2 BACKGROUND

OSs can be defined from several perspectives. However, their primary function is to manage and allocate hardware resources such as memory, processors, and input/output devices, to ensure efficient interactions between users and applications (Tanenbaum, 2009; Silberschatz et al., 2013). An OS abstracts the underlying hardware and resources, providing a user-friendly interface and offering basic services that facilitate interactions for both users and programs.

OSs are generally divided into two categories: general-purpose and special-purpose OSs (Bullynck, 2018). This paper focuses on general-purpose OSs like Windows, Linux, and macOS, designed for broad use and enabling users to organize documents and applications without requiring specific technical expertise. These systems are widely employed in various environments, from households to professional and technical settings, where they support tasks such as document management, billing, and software development. However, general-purpose OSs often fall short when addressing specific domain needs. General-purpose OSs rely exclusively on applications to solve the domain-specific needs of users. For example, household users may find the file and folder structure cumbersome, while profession-

als may struggle to relate documents like bills and orders without relying on billing applications. Technical users may face challenges in organizing their work environments due to limited project-specific support in traditional OSs unless they turn to project management applications. The challenge here is that these applications are vendor-dependent and create a zoo of problems (e.g., file formats, compatibility, interoperability) when integration of multiple tools is needed. These problems can all be solved through the use of "glue" applications (e.g., converters) that increase usage complexity in unnecessary ways (accidental complexity (Brooks, 1987)).

From Model-Based System Engineering (MBSE) perspective (David et al., 2023)(Tolvanen and Kelly, 2016), OSs can be seen as applications that provide specific DSLs for users to model their tasks and interactions. MBSE focuses in the creation of abstract models that represent system architecture, behavior, and interactions leaving aside implementation details (Tolvanen and Kelly, 2016). OSs can be analyzed through their DSLs, which facilitate user interaction, program execution, and hardware management. We decompose OS-DSLs in three level of abstractions: Low-level OS-DSL such as peripheral APIs, allow OEMs to create new devices for computers. Mid-level OS DSLs, such as programming APIs, enable developers to build applications without directly managing hardware. High-level DSLs define elements like files, folders, and windows, enabling visual organization and interaction with the system. In this paper, High-level OS-DSL are our primary focus.

Files and folders are represented visually on the desktop, a limited space defined by the screen area and have attributes like names, sizes, and creation dates, while windows display application content and include attributes such as size, position, and title. This abstraction simplifies user interaction by hiding the underlying complexity. However, traditional OS-DSLs come with several limitations. *Desktops* introduced in the 1970s followed a working environment metaphor and were extended in the 1990s by the concept of *virtual desktops*. Although they provide space for organizing files, they are restricted by the physical screen size, and the icons often lack sufficient visual clarity for smooth navigation. Furthermore, the desktop layout does not persist after a system reboot, requiring users to manually restore their application layouts—a problem recently mitigated in Windows through tools like PowerToy App Layout (Microsoft, 2024). While files and folders are effective for document organization, they often create inconsistencies due to limited modeling degrees (e.g user-defined naming con- ventions and lack of flexibility in organizing dependencies between files). Consequently, handling large volumes of files becomes challenging without advanced organizational tools.

The application-centric nature of traditional OSs presents several challenges. Data re-usability across different applications often requires tedious or complex conversions, and frequent context switching between applications. Resource management is also handled independently by applications, often resulting in redundant or inefficient use of data storage. The heavy reliance on GUIs further limits automation and seamless integration with advanced systems, as many embedded high-level OS-DSLs are not designed for easy programmability (MacOS is an exception here, with its Automator (Apple, 2024) now replaced by the Workflows and Shortcuts app). This limits the ability to automate tasks or integrate with external systems.

In MBSE terms, interacting with an OS through its graphical interface is analogous to modeling. Users create "models" of their desired content and actions through the UI, which the OS interprets and executes, updating the system state accordingly. OS-DSLs abstract underlying complexity, enabling users to focus on tasks without having to deal with low-level system management.

3 HYPERGRAPHOS CONCEPT

HyperGraphOS (HRI-EU, 2024b), as shown in Fig. 2 is designed to redefine how users interact with computers and digital information systems. By leveraging DSLs, HyperGraphOS transforms traditional file management into an interconnected web of information. The basic graphical elements used to create visual DSL are nodes and links. Nodes within the system represent files, documents, and data, which are customizable, annotatable, and maintainable. These nodes visually represent data while encapsulating both content and visual aspects, in a concept similar to Unix symbolic links. Nodes can represent various semantics such as programs, numbers, images, and text, while links define relationships like dependencies, causal connections, and interactions.

The Meta-model allows for the creation of nodes (such as code, documents, and images) and links (such as utilization, realization, and dependency) for users to organize and visualize their data in flexible and dynamic ways. This architecture promotes seamless interaction with various data formats, which are handled by corresponding editors and viewers.

At the heart of HyperGraphOS is the concept of OmniSpace, an infinite network of workspaces, which can be configured with different DSLs, for instance,

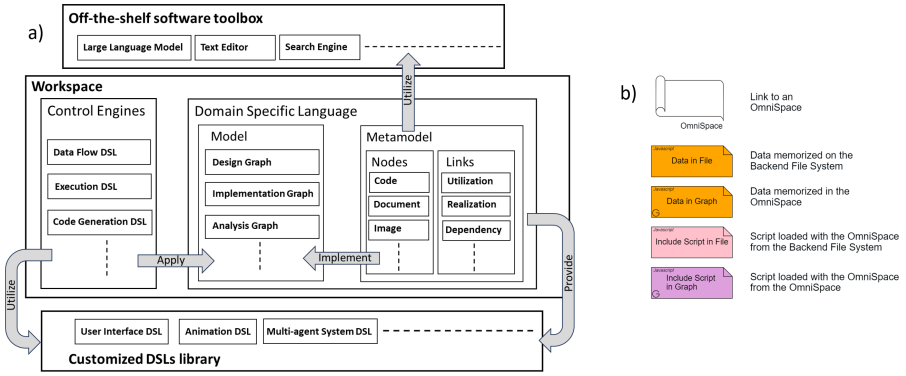

Figure 2: a) HyperGraphOS operation concept. b) Basic DSL for navigation and file manipulation

with DataFlow DSL, Execution DSL, or Code Generation DSL. These DSLs can be used by a developer to model an application which can be executed in-place, on a batch or deployed to a target computer. Execution DSL utilizes the Customized DSLs Library, which includes specialized DSLs such as the Basic DSL (Fig. 2b) for navigation and file manipulation, User Interface DSL, Animation DSL, or AI DSL, providing a rich set of meta-models to developers for their modeling processes.

The Meta-DSL (a specialized DSL to create new DSL) components within HyperGraphOS provides a framework for the development of meta-models, which can be used in different forms, like Design Graphs, Implementation Graphs, and Analysis Graphs. These models use the nodes and links provided by the meta-model, allowing the system to tailor the representation of information to the specific needs of the users. HyperGraphOS offers the capability to define DSLs using Meta-DSLs (notice the recursion here), which are implemented within a set of workspaces for creating and deploying domain-specific languages (a meta-meta model for the creation of DSL is also available to users).

HyperGraphOS operates as distributed workspaces, where users can group documents visually using container nodes, links, and workspace hyperlinks. Workspaces serve as virtual environments for organizing data and applications. They are infinite, flexible, and capable of preserving the state (both model and applications/windows) and navigation history of tasks. Workspaces can also span across multiple storage solutions such as local storage, cloud services, and remote devices. Off-the-shelf Software Toolbox, which includes tools such as LLMs, Text Editors, and Search Engines, are seamlessly integrated into HyperGraphOS. These tools are utilized by the workspace's Control Engines and Meta-model to provide advanced capabilities, such as content creation, search functionalities, and programmatic manipulation.

HyperGraphOS incorporates cutting-edge technologies like an integrated JavaScript-based shell for testing and manipulating nodes and links programmatically. Additionally, a robust search engine and AI integration provide on-demand assistance within documents. Despite its rich set of capabilities, HyperGraphOS maintains a minimalistic design and system footprint, ensuring intuitive interaction and ease of use. The concept of applications, in HyperGraphOS, is re-imagined as modular constructs, moving away from the monolithic executable model of traditional OSs. The set of Featured-Based Code Generation Systems allows for automatic code generation based on models, templates (HRI-EU, 2024b), annotations, ... something that further streamlines software development processes.

Although this is still in development, we strive to implement collaborative features, enabling multi-disciplinary teams to work together using integrated tools for real-time editing, version control, and interactive annotations.

4 SYSTEM ARCHITECTURE

HyperGraphOS is built on a modular architecture as shown in Fig. 3. The architecture is composed of five main modules: a Kernel Interface, a Back-end, Front-ends, External Cloud Services, and Data management. These modules work together to provide a flexible, distributed, and scalable system that redefines traditional file management and work organization. At the core of HyperGraphOS is the Kernel Interface, which manages the hardware abstraction and rendering of the user interface. The Rendering En-

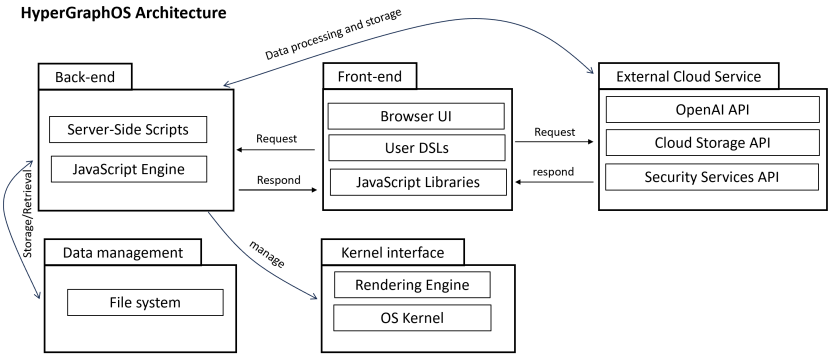

Figure 3: HyperGraphOS software architecture

gine within this module ensures seamless graphical interaction, while the OS Kernel interfaces with essential hardware resources like CPU, memory, and input/output devices. Additionally, the File System plays a crucial role in managing data storage and retrieval, interfacing with the back-end for efficient data processing and handling of user workspaces.

The Back-end acts as an intermediary between the front-end and the kernel, built on JavaScript Engine components and Server-Side Scripts that handle data requests, workspace management, and batch execution. This module processes user requests, manages workspace files as JSON objects, and ensures that data is stored or retrieved from the file system. Moreover, the back-end is responsible for interacting with External Cloud Services, enabling integration with local or cloud-based APIs like OpenAI for AI tasks (Camara et al., 2023), Cloud Storage for scalable data handling, and Security Services for managing data protection and privacy.

The Front-end of HyperGraphOS operates through a browser interface, where users interact with workspace via a graphical canvas powered by user DSLs, JavaScript Libraries and GoJS (one of the most complete graphical libraries available in the web domain (Northwoods, 2022)). This dynamic interface allows users to manipulate visually the content of workspaces, using graphs to represent nodes, and links. Each workspace is stored as a JSON object, allowing light and flexible storage (Nurseitov et al., 2009)(Zunke and D'Souza, 2014) and intuitive management of files and documents. Easy access of its content is possible through a visual inspector or in a programmatic way. The front-end ensures that users have a streamlined, "zero-setup" usability, and interactive experience, directly connected to the back-end for data requests and processing.

The External Cloud Services module integrates key functionalities that extend the capabilities of HyperGraphOS. Through cloud-based APIs, the system interacts with external tools for data processing, security management, and AI-driven features. Services like OpenAI API provide powerful machine learning and natural language processing capabilities used for code generation (Sadik et al., 2023) or dataflow applications, while Cloud Storage offers scalable and distributed storage solutions, ensuring the system can handle an increasing number of workspaces as user needs grow. The Security Services API guarantees user data protection, ensuring privacy and compliance with security standards while maintaining a lightweight system architecture.

Data management in HyperGraphOS is streamlined through a custom organization of files and directories on the server side, bypassing for now the need for traditional databases. This approach simplifies data architecture while ensuring flexibility in managing workspaces. However, as HyperGraphOS evolves, further enhancements may be required to accommodate more complex data management needs.

Scalability is inherently managed through the distributed handling of JSON files that represent workspaces. This ensures that the system can efficiently manage multiple nodes or workspaces without the need for extensive infrastructure, enabling HyperGraphOS to scale seamlessly as users create and manage more complex environments. Security and privacy are handled through external services, allowing HyperGraphOS to maintain a lightweight architecture without sacrificing user data protection. By outsourcing security management to specialized services, the system remains streamlined, ensuring robust data protection without adding unnecessary complexity to the core architecture.

HyperGraphOS also provides robust integration capabilities through its own APIs, allowing programmatic navigation and modification of workspace models, with models represented in a dual way as a visual drawing and as a JSON object. JSON format has been chosen for its native integration in Javascript

and its relative lightweight encoding (Nurseitov et al., 2009)(Zunke and D'Souza, 2014). Besides JSON, some dedicated user interface and DSLs make use of the YAML format (Eriksson and Hallberg, 2011) for its compactness, readability and user friendliness. Both the front-end and back-end offer libraries that support users in defining code generators for their applications, making integration with external tools and systems straightforward and seamless.

This modular and distributed architecture, combined with the flexibility offered by JSON-based workspace management, allows HyperGraphOS to deliver a scalable solution for modern computing needs. It offers a new approach to manage digital information by extending high-level OS-DSLs, blending the simplicity of user-friendly design with the power of advanced, customizable back-end services.

5 CASE STUDIES

In this section, three case studies are presented to demonstrate the practical application of various system modeling and artificial intelligence methodologies using HyperGraphOS. Each case study highlights the significant contribution that HyperGraphOS provides in the creation of a multi-agent robotic task execution system, a meta-model for system architectures in research applications, and a dialog management system. The first two will be briefly presented, and the last one will be explored in greater detail.

Case Study 1: Multi-Agent Robotic Task Planning and Execution

In this case study, HyperGraphOS is used to develop a robotic control system based on multi-agent task planning and execution (Joublin et al., 2024b). The system integrates natural language processing with task and motion planning using a hierarchical architecture built with OpenAI's LLMs. The CoPAL (Cognitive Planning and Learning) system allows the robot to perform complex tasks in the real world, such as preparing pizza and stacking cubes. This is achieved by incorporating replanning feedback loops. The model, defined using the dataflow DSL, integrates components allowing ROS (Quigley et al., 2009) communication with the robotic system. The research and development of CoPAL with HyperGraphOS demonstrates the flexibility of modeling, executing, debugging, and testing complex data flow models that generate tasks for a humanoid robot in the real world. The core development of the multi-agent system, including the DSL, took only a single week, allowing most of the time to be spent on evaluation and experiments with the model.

In continuation of this work, HypergraphOS is now used for creating LLM-based multiagent system (MAS) architectures. A DSL for agents, multi-agent dialogue arbitration, working rooms and LLM accessible tools are currently the subject of implementation and research. This has been successfully demonstrated by flexible multi-party dialog generation and their analysis using a method also implemented on HypergraphOS (Ebubechukwu et al., 2025).

Case Study 2: Modeling Research Projects with Thebes DSL

This case study addresses the challenge of managing dynamic research projects using Thebes, a lightweight DSL tailored for modeling research projects within HyperGraphOS. Thebes facilitates rapid prototyping and incremental design, enabling seamless integration with existing tools via code generation. Applied to projects like the tabletop robot Haru (Gomez et al., 2018) and the CoPAL system (Joublin et al., 2024b), Thebes significantly improved collaboration and adaptability. HyperGraphOS provided support for creating the metamodel in about 30 minutes and implementing the model integrity checkers and code generation in JavaScript in about three days.

Case Study 3: Virtual Receptionist for Visitor Registration

This case study focuses on the development of a virtual receptionist system used for visitor registration at a research institute (Joublin et al., 2024a). The system, initially developed before the widespread adoption of LLMs, utilized a recursive neural network to define a behavior engine for the AI-driven receptionist. This research explored the challenges of creating dialogue systems capable of interacting with users through natural language or speech. Traditional dialogue systems often face several issues, including the need for extensive training data and difficulties in defining reward functions. They also struggle with limited control and explainability.

To address these challenges, the team designed a neural behavior engine inspired by neurobiology and neuropsychology, which incorporated concepts such as mirror neurons and multi-modal embodiment. This engine facilitated mixed-initiative dialog and action generation. The system was successfully implemented as a virtual receptionist in a semi-public space, demonstrating its capability to manage real-world interactions with users.

HyperGraphOS played a pivotal role in two key aspects of the system's development:

- HyperGraphOS was used to define a DSL for the

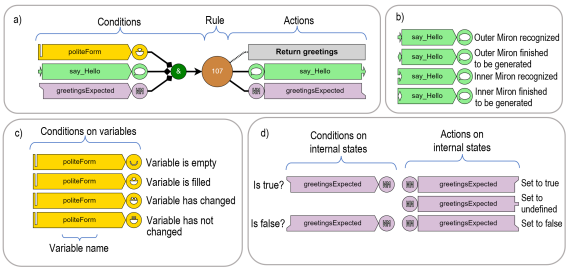

Figure 4: Rule DSL Elements : A) Graphical representation of a rule defined by its auto-generated ID (107), its conditions and its actions. The gray box is just a comment used to explain the rule. B) Possible state of a Miron (an abstraction used to equally recognize and generate sentences) that can be used as part of a condition. C) Possible state of a variable that can be used as part of a condition. D) Possible conditions and actions on internal states. Link visual aspect is automatically determined by the connected nodes.

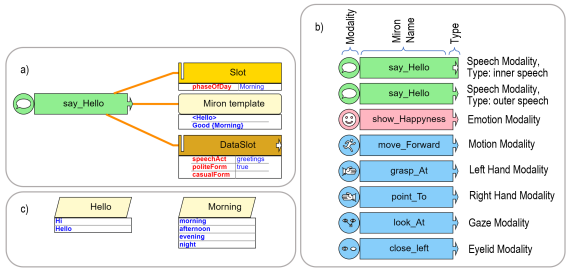

Figure 5: Miron DSL elements: A) Graphical representation of a Miron defined by a modality, a name, a type (inner or outer), templates, named entities (slots) and associated data (data slots). B) Example of different Miron modalities. Modalities were used to control speech output and motion and expression of a virtual avatar. C) Grammar fields defining alternative verbal expressions.

behavior engine together with a code generator. The DSL was created in 3 days and the code generator in one week. The DSL is based on a clock-based architecture model created in a dedicated workspace, and the generated code integrated into a target JavaScript module for the avatar receptionist system.

- HyperGraphOS was also employed to design a Dialog DSL (Fig. 4 and 5) based on parallel state flow. The DSL and the code generation (Fig. 6) has been created in about two weeks. The model, implemented in a workspace, consisted of 4246 nodes and 3890 links, which generate JavaScript files such as dictionaries (4033 generated lines), weights for the recurrent network (4659 generated lines), and NLP intents (2410 generated lines). Average code generation time (from reading the model to generating all files) was less than 3 seconds on a 12th Gen Intel Core i7-12700H laptop.

To illustrate the development of a DSL and the

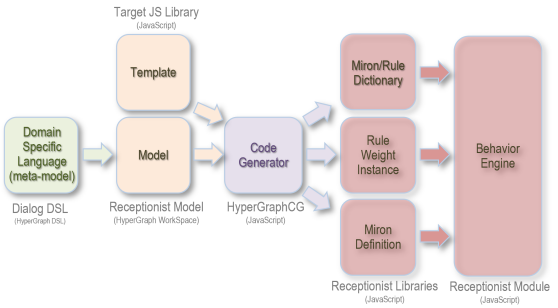

Figure 6: Avatar Receptionist Network Modeling Process

```
//============================
// GetRuleList
//
// THIS CODE IS GENERATED FROM:
//[# Define FileName #]
//:// %b
// ../../../apps/AppReceptionist/2.0/AvatarReceptionist.json
// PLEASE DO NOT EDIT BY HAND
//[# Begin Header Info #]
//:// VERSION: %b
//:// GENERATED: %b
// VERSION:   1.0
// GENERATED: 2021-05-28
//[# End Header Info #]
//============================

//[# Define Avatar #][LinePattern]
//:function getRuleList_%b() {
 function getRuleList_Receptionist() {
   return [
     ////////////////////
     // Define Rules
     ////////////////////
     //[# Loop Begin Rule #]
     //[# Begin Header Rule #]
     //:{ // RULE %b (%b): %b
     //:  name:  '%b',
     //:  index: %b,
     { // RULE 7 (n4:n723): MEISY ----> Dialog end
       name: 'rule7',
       index: 7,
       //[# End Header Rule #]
       fanIn: [
                //[# Loop Begin FanIn #]
                //[# Begin OR #][ArrayPattern]
                //:[${ src: '%i', %tmironName: '%i', %ttype: '%i', %tweight: %i },   %t// %i
                //:  { src: '%i', %tmironName: '%i', %ttype: '%i', %tweight: %i }$], %t// %i
                [ { src: 'previousRule',  mironName: 'rule6',
                  { src: 'actionDone',    mironName: 'request_EndApplication',
                //[# End OR #]
                //[# Loop End FanIn #]
                //[# Begin Skip #]
                [{ src: 'actionDone',     mironName: 'wait_Timeout{timeout:180,id:6}',
                  { src: 'previousRule',  mironName: 'rule6',
                //[# End Skip #]
                ],
       //[# Begin FanOut #][ArrayPattern]
```

Figure 7: Example of Avatar Dialog Network source template. Comments like '//[# command #] represent code generation commands, while comments like '//: ...' represent command parameters.

process of code generation in HyperGraphOS, this case study focuses on the creation of the Dialog DSL (Fig. 9). This case study followed the complete DSL creation process in HyperGraphOS to define a specific DSL. The first step involved defining a meta-meta-model to determine the visual representation of each DSL element. HyperGraphOS supports this phase with JavaScript functions leveraging the GoJS model concept. Once the meta-meta-model was established, the meta-model for the Dialog DSL was defined, which can be visually designed within a workspace using HyperGraphOS's dedicated DSL for building DSLs. During this stage, the elements of the Dialog DSL were specified along with their attributes and semantics (see Fig. 4 and 5). HyperGraphOS automatically adds the user-defined DSL to a system palette for easy access.

Currently, HyperGraphOS offers two primary approaches for defining DSLs: 1) a full process that involves creating both the meta-meta-model (using JavaScript and GoJS) and the meta-model (drawn in

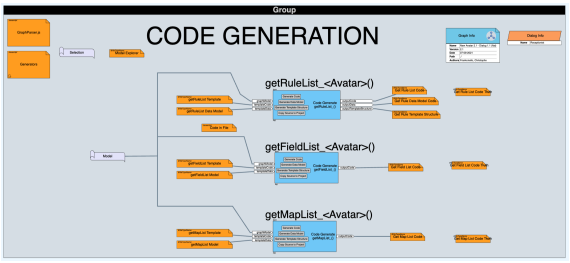

Figure 8: Code generation model for the Dialog Model

workspace), and 2) a light process, where users define a meta-model by parameterizing a DSL creation tool within workspaces. Additional methods for defining DSLs are under exploration and will be addressed in future publications.

For code generation in the Dialog Model created for this application, one of HyperGraphOS's template engines was used. HyperGraphOS provides several template generators, which can be extended by users. The process involves the following steps: first, a target example file is required, serving as a running example of the code to be generated. The example file is then transformed into a template by adding annotations as comments (see Fig.7). Next, the code generation logic is defined in a model (see Fig.8), where it is implemented.

The system demonstrated robustness in real-time scenarios, effectively managing dialog states and context, and seamlessly switching between different modalities (e.g., speech or text interaction, or a combination of both, including telephony) while gracefully handling errors. The use of DSLs for behavior and engine modeling facilitated scalability and maintainability, ensuring ease of maintenance throughout the development and testing process. In particular, the code generation of rules produced files describing neural network weights, which drastically streamlined the creation process that would have been too complex and error prone manually.

6 DISCUSSION, CONCLUSION, AND FUTURE WORK

6.1 DISCUSSION

HyperGraphOS marks a major advancement in OSs, tailored for complex applications. Leveraging a web-based architecture and DSLs, HyperGraphOS offers an adaptive and robust platform for managing complex models and data. This section compares HyperGraphOS to other state-of-the-art systems and highlights its unique contributions.

In comparison to systems like PlantUML (Correia et al., 2024) and Graphviz (Gansner, 2009), which are widely used for static diagram creation and visualization, HyperGraphOS distinguishes itself by enabling dynamic interactions with graph models. The ability to manipulate nodes and links programmatically using JavaScript and navigate virtually OmniSpaces sets HyperGraphOS apart from traditional graph modeling tools.

When compared to DSL-centric systems like MetaEdit+ (Tolvanen and Kelly, 2016), JetBrains MPS (Pech et al., 2013), and Eclipse Xtext (Herrera, 2014), HyperGraphOS provides a more intuitive and accessible interface due to its web-based architecture and extensive use of the visual workspace. Its simplicity in creating and adapting DSLs enables rapid prototyping and incremental development, which is especially beneficial for dynamic research projects.

HyperGraphOS and WebGME (Maróti et al., 2014) represent two distinct approaches to graph-based and meta-modeling environments. WebGME excels in its collaborative, web-based infrastructure designed to create and manage Domain-Specific Modeling Languages (DSMLs), featuring scalable version control and prototypical inheritance to unify metamodeling and modeling. HyperGraphOS, in contrast, operates as a meta-operating system that integrates graph-based structures with Domain-Specific Languages (DSLs) and advanced AI capabilities. Its workspace design allows for infinite, interconnected workspaces, enabling design-time execution. While WebGME focuses on multi-paradigm modeling with rigorous collaboration mechanisms, HyperGraphOS stands out with its dynamic execution environment and adaptability for rapidly evolving interdisciplinary applications.

HyperGraphOS and jjodel (Di Rocco et al., 2023) both aim to enhance model-driven engineering by simplifying complexity and providing advanced modeling capabilities, yet their approaches differ significantly. Jjodel is a cloud-based reflective modeling framework designed for simplicity and real-time collaboration. It emphasizes accessibility with intuitive syntax customization and geometry-based notation, particularly beneficial in technical domains like railways. HyperGraphOS, on the other hand, introduces a unique meta-operating system architecture, supporting complex workflows through a network of workspaces and seamless integration of AI-driven modeling and execution tools. While jjodel focuses on educational and lightweight collaborative modeling, HyperGraphOS targets interdisciplinary and computationally intensive scenarios.

Sirius Web (Giraudet et al., 2024) is a web-based

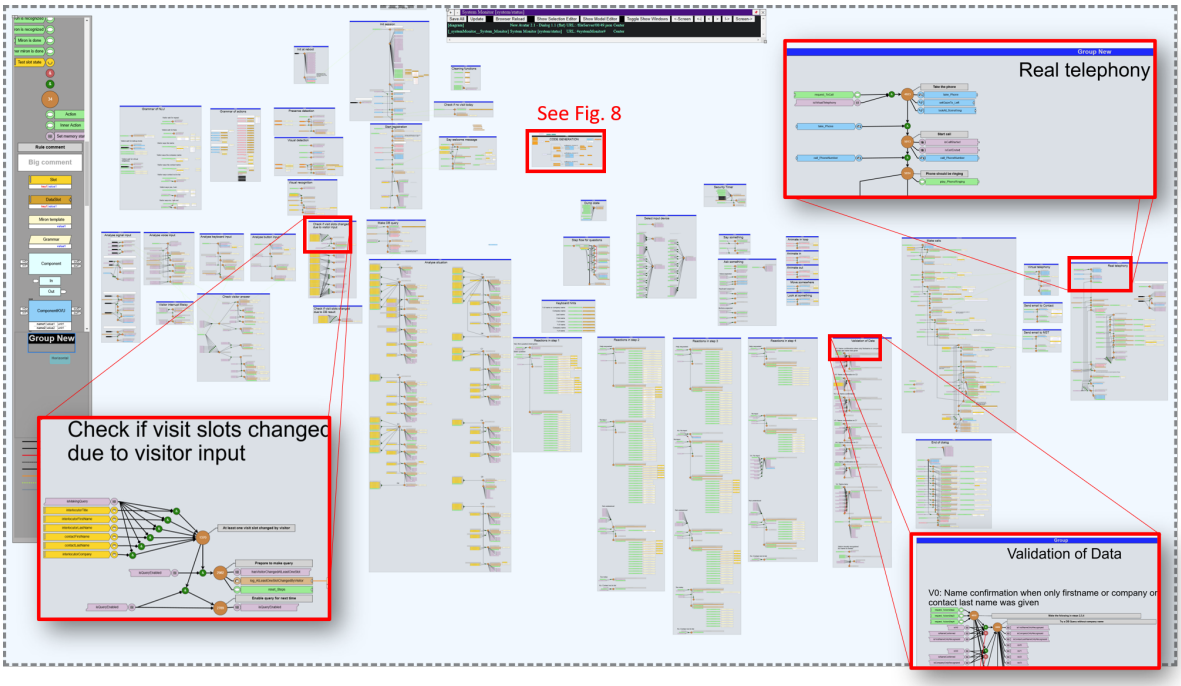

Figure 9: View of the full Dialog Model for the Avatar Receptionist. The code generation model shown in figure 8 is in the center top of this model

language workbench tailored for developing graphical Domain-Specific Modeling Languages (DSMLs) and their environments. Built on the lessons from Sirius Desktop, Sirius Web supports collaborative modeling, offers predefined representation types such as diagrams and Gantt charts, and provides both low-code and API-driven interfaces for studio creation. While HyperGraphOS excels in versatility and integration for complex, adaptive systems through its graph-based and AI-augmented architecture, Sirius Web specializes in facilitating the development and deployment of graphical DSMLs with a strong emphasis on user experience and streamlined workflows.

6.2 CONCLUSION

In this paper, we presented HyperGraphOS, a modern DSL-based OS designed specifically for scientific and engineering applications. Through the use of meta-meta modelling, DSLs and graph-based model representations, HyperGraphOS provides users with an intuitive and flexible platform for creating, manipulating, and visualizing complex models and data. HyperGraphOS's open-source nature invites further exploration and contributions from the community. The tool is available as open-source software and can be accessed at (HRI-EU, 2024a), where additional documentation and videos (HRI-EU, 2024b) and future updates are posted.

The case studies in robotic task planning, dynamic research projects, and virtual receptionist systems demonstrate HyperGraphOS's versatility and practical benefits. In comparing HyperGraphOS to other state-of-the-art systems, its important contributions were outlined, such as dynamic graph model interaction, openness in creating new DSLs, and integration with AI components. While HyperGraphOS presents numerous advantages, it also opens up opportunities for further enhancements, particularly in data handling, scalability, and facilitating collaboration. Expanding these capabilities will be essential as the system evolves to handle increasingly complex and larger applications.

6.3 FUTURE WORK

Moving forward, there are several areas for improvement in HyperGraphOS. While relying on external services for security and privacy management offers flexibility, future iterations could include robust, built-in security measures to strengthen data protection. As the system scales to support larger and more complex projects, enhancing performance while maintaining a user experience quality will be essential.

HyperGraphOS has the potential to supports

AMDD by introducing AI-powered modeling that enhances productivity (Sadik et al., 2024). Users benefit from AI-generated suggestions and improvements during the code generation phase. Collaboration is still under-developed, and functionalities to enable multi-disciplinary teams to work together using integrated tools for real-time editing, version control, and interactive annotations have been started. These functions would help fostering synergy among team members from various domains. A drawback of workspace is its reliance on large screens for comfortable use. One potential mitigation could involve enabling the use of HyperGraphOS in Virtual Reality settings. HyperGraphOS shows great potential but requires further development in several areas.

Today, most operating systems evolved towards a common paradigm, making them nearly interchangeable. We propose that a more fruitful avenue lies in the development of Operating Systems that cater to the diverse needs of users. For instance, the recent proposal of Space OS by the startup Deta (Pierce, 2023) presents a new cloud-based operating system aimed at redefining personal computing. It gives users control over their applications and data in a personal cloud, offering a flexible and customizable environment where users can manage apps from any device. Additionally, the platform incorporates AI-powered tools like Teletype, enabling non-programmers to build applications through natural language. Another example is the Rabbit R1 (Technology, 2024), a compact AI-powered device that simplifies user interaction with apps and services. It features a 2.88-inch touchscreen, a 360-degree rotating camera, and runs Rabbit OS, an AI-driven operating system. The R1 functions as a personal assistant, capable of performing tasks such as booking flights, playing music, or ordering food, all through voice commands. Powered by its Large Action Model (LAM), it provides faster responses and personalized functionality, positioning it as a next-generation alternative to traditional smartphone interfaces.

REFERENCES

Apple (2024). Automator user guide for macos. `https://support.apple.com/guide/automator/welcome/mac`. Accessed: 22-Sep-2024.

Blair, G., Coupaye, T., and Stefani, J.-B. (2009). Component-based architecture: the fractal initiative. *annals of telecommunications-annales des telecommunications*, 64:1–4.

Brooks, F. P. (1987). No silver bullet: Essence and accident of software engineering. *Computer*, 20(4):10–19.

Bullynck, M. (2018). What is an operating system? a historical investigation (1954–1964). *Reflections on programming systems: Historical and philosophical aspects*, pages 49–79.

Camara, J., Troya, J., Burgueno, L., and Vallecillo, A. (2023). On the assessment of generative ai in modeling tasks: An experience report with chatgpt and uml. *Software and Systems Modeling*, 22(3):781–793.

Correia, F. F., Ferreira, R., Queiroz, P. G., Nunes, H., Barra, M., and Figueiredo, D. (2024). Towards living software architecture diagrams. *arXiv preprint arXiv:2407.17990*.

David, I., Latifaj, M., Pietron, J., Zhang, W., Ciccozzi, F., Malavolta, I., Raschke, A., Steghofer, J., and Hebig, R. (2023). Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. *Software and Systems Modeling*, 22(1):415–447.

Di Rocco, J., Di Ruscio, D., Di Salle, A., Di Vincenzo, D., Pierantonio, A., and Tinella, G. (2023). Jjodel–a reflective cloud-based modeling framework. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 55–59. IEEE.

Ebubechukwu, I., Ceravola, A., Joublin, F., and Tanti, M. (2025). Automating dialogue evaluation: Llms vs human judgment. In *Proceedings of the 27th International Conference on Human-Computer Interaction (HCI International 2025)*, Gothenburg, Sweden. Human-Computer Interaction International, Springer. Accepted on December 9, 2024.

Ediz, Ö. (2009). "improvising" architecture: A fractal based approach. In *Computation: the new realm of architectural design: 27th eCAADe Conference proceedings*, pages 593–598.

Eriksson, M. and Hallberg, V. (2011). Comparison between json and yaml for data serialization. *The School of Computer Science and Engineering Royal Institute of Technology*, pages 1–25.

Gansner, E. R. (2009). Drawing graphs with graphviz. *Technical report, AT&T Bell Laboratories, Murray, Tech. Rep, Tech. Rep.*

Giraudet, T., Bats, M., Blouin, A., Combemale, B., and David, P.-C. (2024). Sirius web: Insights in language workbenches-an experience report. *The Journal of Object Technology*.

Gomez, R., Szapiro, D., Galindo, K., and Nakamura, K. (2018). Haru: Hardware design of an experimental tabletop robot assistant. In *Proceedings of the 2018 ACM/IEEE international conference on human-robot interaction*, pages 233–240.

Herrera, A. S.-B. (2014). Enhancing xtext for general purpose languages. In *MoDELS (Doctoral Symposium)*.

HRI-EU (2024a). Hypergraphos. `https://github.com/HRI-EU/hypergraphos`. Accessed: 21-Oct-2024.

HRI-EU (2024b). Hypergraphos-documentation. `https://github.com/HRI-EU/hypergraphos/`

tree/main/Documentation/Videos. Accessed: 22-Oct-2024.

Joublin, F., Ceravola, A., and Sandu, C. (2024a). Introducing brain-like concepts to embodied handcrafted dialog management system. *arXiv preprint arXiv:2406.08996*.

Joublin, F., Ceravola, A., Smirnov, P., Ocker, F., Deigmoeller, J., Belardinelli, A., Wang, C., Hasler, S., Tanneberg, D., and Gienger, M. (2024b). Copal: corrective planning of robot actions with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8664–8670. IEEE.

Lorenz, W. E. (2002). *Fractals and fractal architecture*. na.

Maróti, M., Kecskés, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., Levendovszky, T., and Lédeczi, Á. (2014). Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure. *MPM@ MoDELS*, 1237:41–60.

Microsoft (2024). Powertoys for windows. https://learn.microsoft.com/en-us/windows/powertoys/. Accessed: 22-Sep-2024.

Northwoods (2022). Gojs: Powerful diagrams for every industry. https://gojs.net/latest/index.html. Accessed: September 2024.

Nurseitov, N., Paulson, M., Reynolds, R., and Izurieta, C. (2009). Comparison of json and xml data interchange formats: a case study. *Caine*, 9:157–162.

Pech, V., Shatalin, A., and Voelter, M. (2013). Jetbrains mps as a tool for extending java. In *Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools*, pages 165–168.

Pierce, D. (2023). Space os is trying to bring back the personal computer — this time in the cloud. https://www.theverge.com/2023/10/10/23911145/space-os-deta-personal-cloud-computer. Accessed: 21-Oct-2024.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan.

Sadik, A. R., Brulin, S., and Olhofer, M. (2024). Coding by design: Gpt-4 empowers agile model driven development. In *The International Conference on Model-Based Software and Systems Engineering - MODELSWARD 2024*, pages 149–156.

Sadik, A. R., Ceravola, A., Joublin, F., and Patra, J. (2023). Analysis of chatgpt on source code. *arXiv preprint arXiv:2306.00597*.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2013). *Operating system concepts essentials*. Wiley Publishing.

Tanenbaum, A. (2009). *Modern operating systems*. Pearson Education, Inc.,.

Technology, R. (2024). Rabbit r1 - ai-powered personal assistant. https://www.rabbit.tech/rabbit-r1. Accessed: 22-Sep-2024.

Tolvanen, J.-P. and Kelly, S. (2016). Model-driven development challenges and solutions: Experiences with domain-specific modelling in industry. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 711–719. IEEE.

Zunke, S. and D'Souza, V. (2014). Json vs xml: A comparative performance analysis of data exchange formats. *IJCSN International Journal of Computer Science and Network*, 3(4):257–261.