UNIVERSITÄT TRIER

Semantic-Aware Coordinated Multiple Views for the Interactive Analysis of Neural Activity Data

Vom Fachbereich IV der Universität Trier zur Verleihung des akademischen Grades Doktor der Naturwissenschaften (Dr. rer. nat.) genehmigte Dissertation

von

Christian Nowke

Trier, 2024

Betreuer*in: Jun.-Prof. Dr. Benjamin Weyers
1. Berichterstatter*in: Jun.-Prof. Dr. Benjamin Weyers
2. Berichterstatter*in: Prof. Dr. Torsten W. Kuhlen
Datum der Disputation: 22.05.2023

Abstract

Visualizing brain simulation data is in many aspects a challenging task. For one, data used in brain simulations and the resulting datasets is heterogeneous and insight is derived by relating all different kinds of it. Second, the analysis process is rapidly changing while creating hypotheses about the results. Third, the scale of data entities in these heterogeneous datasets is manifold, reaching from single neurons to brain areas interconnecting millions. Fourth, the heterogeneous data consists of a variety of modalities, e.g.: from time series data to connectivity data, from single parameters to a set of parameters spanning parameter spaces with multiple possible and biological meaningful solutions; from geometrical data to hierarchies and textual descriptions, all on mostly different scales. Fifth, visualizing includes finding suitable representations and providing real-time interaction while supporting varying analysis workflows. To this end, this thesis presents a scalable and flexible software architecture for visualizing, integrating and interacting with brain simulations data. The scalability and flexibility is achieved by interconnected services forming in a series of Coordinated Multiple View (CMV) systems. Multiple use cases are presented, introducing views leveraging this architecture, extending its ecosystem and resulting in a Problem Solving Environment (PSE) from which custom-tailored CMV systems can be build. The construction of such CMV system is assisted by semantic reasoning hence the term *semantic-aware* CMVs.

Zusammenfassung

Das Visualisieren von Simulationsdaten des Gehirns ist eine Herausforderung. Zum einen sind die zur Simulation herangezogenen Daten sowie die Simulationsergebnisse heterogen, und Erkenntnis wird durch ein Verknüpfen der Daten gewonnen. Des Weiteren unterliegt der Analyseprozess ständigen Veränderungen, während Hypothesen abgeleitet werden. Auch sind die Skalen der in den Simulationsdatensätzen enthaltenen Entitäten vielfältig: Vom einzelnen Neuron bis hin zu Gehirnarealen mit Millionen unter sich verbundenen Neuronen. Ferner bestehen die Daten aus unterschiedlichen Modalitäten, z.B.: Sie reichen von Zeitserien bis hin zu Konnektivitätsdaten, von einzelnen Parametern zu einer Menge an Parametern, die einen Parameterraum aufspannen, der viele und biologisch bedeutende Lösungen enthalten kann; von geometrischen Daten zu Hierarchien und textuellen Beschreibungen, alle zumeist in unterschiedlichen Skaleneinheiten. Letztlich beinhaltet das Visualisieren auch das Finden geeigneter Repräsentationen und das Bereitstellen von echtzeitfähigen Interaktionen, wobei auch unterschiedlichste Analyseabläufe unterstützt werden sollen. Daher präsentiert diese Doktorarbeit eine skalierbare und flexible Softwarearchitektur zur Visualisierung, Integration und Interaktion mit Gehirnsimulationsdaten. Die Skalierbarkeit und Flexibilität wird durch miteinander verbundene Diensten erreicht, die zusammen ein Coordinated Multiple View (CMV) System bilden. Verschiedene Anwendungsfälle werden vorgestellt, die Ansichten bereitstellen und diese Architektur nutzen sowie erweitern, sodass ein Ökosystem entsteht, welches eine Problemlösungsumgebung schafft, aus der spezifische, auf die Bedürfnisse zugeschnittene CMV Systeme abgeleitet werden. Die Konstruktion dieser CMV Systeme wird durch computergestütztes semantisches Deduzieren assistiert, daher der Begriff *semantisch bewusste* CMVs.

Acknowledgments

A special thanks goes to Professor Dr. Benjamin Weyers. Your support and motivation have helped me a lot throughout this journey. Thank you for your eager willingness to proof read the manuscripts and your endless input on ideas, concepts, and suggestions upon improving this work. It was much very appreciated!

I would also like to thank Professor Dr. Torsten W. Kuhlen, for his encouragement and guidance throughout my time at the Virtual Reality Group. Thank you for this opportunity and invaluable advice that made this thesis possible.

In addition, I would like to express my thanks to Dr. Bernd Hentschel for his witty remarks, his emphasize to always remember that writing a thesis is a marathon and not a sprint, and finally his background assistance so that we all could keep doing what we love. And of course, to all my colleagues at the Virtual Reality Group for all the interesting discussions and fond memories that will always remain with me.

Also, I want to express my deepest gratitude to my beloved wife. Dear Alexa, without your endless support and care of almost all secular things that fill our daily life, this thesis would not have been possible. Thank you so much!

Finally, I cannot thank enough my parents that always have been there for me. Back in the days when life was so much simpler, as a young want-to-be-adult, you were never short of advice, albeit often just ignored to eventually see its wisdom later on. I will never forget how easy it was—without exception—to persuade my father buying the next cutting edge overpriced book in computer science that I wanted to read. And to my mother, without her I would definitely had starved to death over the course of studying. Thank you all so much, I deeply love you all!

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

The human brain is one of the most complex biological systems that is known to mankind. Discovering the principles of its modus operandi is a research goal already spanning several decades, deepening our understanding of its structure and functioning [Bassett & Gazzaniga, 2011]. Large-scale research projects like the Human Brain Project[1] or the American Brain Initiative[2] were initiated to focus and foster interdisciplinary research in this area. The introduction of computer systems in the last century and their ever increasing availability to a broad audience allows for the exploration of a fundamental question already posed by Neumann [1958]: is it possible to mimic an *information processing system* such as the brain by the means of computer simulations?

This question originated the field of *computational neuroscience*, a relatively new and very active research discipline. Computational neuroscience aims to derive mathematical models for the governing principles of the brain through observations of data from biology and by application of these models as simulations on specific hardware platforms [Sejnowski *et al.*, 1988]. The basic, underlying concept of any given neural simulation is a model, formulated in mathematical terms, describing the time-varying dynamics of neurons and their connections to other ones. A neuron is a small, however complex, biological system processing information through electrical signals. Mathematical models, consisting of large interconnected neurons, vary in degree of abstraction, precision, and approximation of behavior [Hodgkin & Huxley, 1952; Izhikevich, 2004; Stein, 1967]. Based on these, simulations are performed by computing the dynamics of neurons massively in parallel. The overall goal of simulating large-scale neural networks is the development of models that reproduce and therefore mimic certain functionality of the brain [Sejnowski *et al.*, 1988]. To only name a few possible application areas, this knowledge could potentially lead to vast superior classes of autonomous systems, compared to today's capabilities, enabling systems that are fault tolerant to hardware defects or highly accurate data classifiers running as out-of-the-box services [Frost, 2010; LeCun *et al.*, 2015].

A significant advantage of simulating biological systems in comparison to *in vivo* exper-

[1] `https://www.humanbrainproject.eu/`
[2] `https://www.braininitiative.nih.gov/`

iments, is the ability of observing the entire system at any given time. In vivo experiments usually suffer from poor recording fidelity and are restricted by ethical constraints. Yet, simulations introduce a different challenge: in order to converge neural models more closely to their biological counterparts, the amount of data to investigate increases. To this end, *interactive exploratory data visualization and analysis* is becoming increasingly important to gain further insight and convey findings to researchers [Keim *et al.*, 2008]. In a scientific workflow, the visualization of data plays a fundamental role in obtaining knowledge by providing necessary working tools to explore, generate and verify hypotheses about the data [Keim *et al.*, 2008]. An interactive visual analysis facilitates a *user-centric* approach, aiming at extracting insight from data by transforming it into visual representations while on top offering interaction [Shneiderman, 1996; Spence, 2001; Tufte, 1991]. The key idea is to leverage the distinct human capability for pattern recognition in visual perception.

While modeling neural networks is a complex task, the conception of visualizations, offering the aforementioned capabilities, includes finding suitable transformation of data to visual representations and providing real-time interaction with these is a difficult endeavor on its own. To this end, this thesis will explore how interactive visualization using CMVs enriched by a semantic model can aid in the process of interpreting and analyzing neural network simulation results. Thus, I will present a scalable and flexible software architecture for visualizing, integrating, and interacting with data resulting from simulations. The scalability and flexibility will be achieved by loosely coupled independent *services* that are interconnected according to the specific workflow needs. Whenever a new workflow is encountered, new services can easily be added, and existing ones reused by interconnecting them.

1.2 Requirements

Modeling neural networks and the simulation thereof produces heterogeneous, potentially time-varying output, e.g., spike data from neurons, connectivity distributions among neurons, derived statistical observables like mean activity rate and many more, which need analysis in order to form an understanding of the dynamics exhibited by these networks. Moreover, analysis cannot be performed in isolation by only considering one dataset at a time. Researchers need the means to relate simulation output to reveal interesting features hidden in the data.

To this end, data management is a central requirement for any visualization system. In addition, the means to visually analyze simulation data requires finding appropriate visual transformations and interactions. On top, supplemental data are derived and aggregated from simulations, e.g., mean firing rate and connectivity for sets of neurons, etc., which need to be visualized, accessed and stored.

To address these emerging heterogeneous datasets and the resulting scientific questions, distinct visualization techniques are required to extract insight. However, the most appropriate visualization design depends on the specific research question. Thus, it is possible that for even the same dataset, multiple visualization designs coexist emphasizing certain features in the data best. Because researchers need to relate data, visualization designs must be concurrently accessible and it must be possible to share interaction among them. A standard visualization technique to handle such a requirement is termed CMVs [North

& Shneiderman, 1997; Roberts, 2007; Wang Baldonado *et al.*, 2000]. CMVs can combine spatial and non-spatial data depictions into *one visualization design space*. However, classical CMV systems are restricted to run on a single computer, limiting the choice of display systems for the visual design space and by this restriction may neglect the best way to convey insight. For instance, spatial data can benefit from *immersive visualization* techniques which facilitate a user-centered projection to improve depth perception [Laha *et al.*, 2012; Raja *et al.*, 2004; Ware & Franck, 1994]. Restricting the visual design space to non-immersive display systems prohibits to leverage this benefit. But many data resulting from neural simulations are not necessarily spatial: a function plot of the mean activity for a set of neurons is inherently two dimensional. Here, a traditional 2D User Interface (UI) is usually preferred. Ideally, a CMV system allows for distributing and linking visualizations over system boundaries to use as many and as diverse display systems as needed.

Moreover, exploratory data analysis can rapidly shift focus due to new hypotheses or a discovery, resulting in new requirements for the analysis, exploration, and visualization. This makes it hard to define standard workflows for analyzing neural network simulations. Hence, a visualization system has to cope with changing workflows. To this end, the extensibility of such a visualization system is a requirement, precluding a monolithic software design. The extensibility is twofold: firstly, the system must make it easy to add and link visualizations. Secondly, researchers need to define how visualizations for a workflow are linked and coordinated, ideally in form of a scriptable Problem Solving Environment (PSE). PSEs are a class of software systems that support algorithm development, performance tuning, and application steering for scientific exploration and visualization [Parker *et al.*, 1998]. Hence, this PSE must allow for a distributed CMV system to expose functionality, assist in the invocation of visualizations, and provide the means to define coordination among views. To expose a coordinated view's functionality in a PSE and specify coordination, a *semantic model* is helpful. A CMV system enriched by semantic model is termed a *semantic-aware* CMV system in context of this work.

A further requirement for a semantic-aware CMV system is the integration of existing community tools. It should close the gap between experimental and computational neuroscience by leveraging existing statistical evaluation frameworks developed in both subdomains.

In summary, the requirements for such a system are:

1. integrate and manage heterogeneous datasets.
2. provide analysis capabilities by conceiving visualizations and interactions with the depicted data.
3. visualize and manage supplemental data aggregated after the simulation concludes.
4. enable the development of a scalable, flexible, and extensible visualization system applying CMVs.
5. support immersive and non-immersive display system for the design space of visualizations.
6. describe functionality by a semantic model to assist in the composition of functional units and the definition of coordination among views inside a PSE.

7. provide the means to integrate existing tools from the neuroscience domain.

The next section will discuss the thesis's contributions to develop this semantic-aware CMV system.

1.3 Contributions

The central contribution of this thesis is the derivation, the development, and the implementation of an applicable semantic-aware CMV system. To this end, a CMV system, based on a loosely coupled Event-Driven Architecture (EDA) that uses semantic reasoning to *dynamically* specify coordination is presented that integrates various visualizations. Its applicability will be shown along three use cases in neuroscience that enable the exploration of neural activity data resulting from simulations. The first use case centers around the investigation of neural activity data from a large-scale spiking neural network and presents novel visualization techniques for the available data by introducing views and coordination endpoints for a CMV system. The second use case focuses on the visualization and integration of statistical analysis for spike trains. The final use case presents an approach to visualize and steer a structural plasticity simulation utilizing the communication infrastructure developed for the PSE. All use cases leverage the presented architecture. Components developed as part of one use case are reused in different ones, exploiting the benefits of a loosely coupled EDA.

The presented CMV systems ease the development of novel visualizations in Hybrid Virtual Environments (HVEs) where users can dynamically coordinate views across display devices to utilize one visual design space. In summary, the following contributions are made:

- a *communication infrastructure* is developed, based on principles of EDAs, allowing for simple event definitions.
- a *semantic model* for coupling CMV systems is presented to define workflows, establish coordination among views, and model dependencies.
- a *runtime system* that manages the invocation of views and their coordination based on workflows.
- several custom-tailored views forming specialized CMV systems for each presented use cases.

The communication infrastructure will allow for the definition of EDA *services* and data exchange via events. Its main focus is the extensibility to add new events to integrate views for the CMV system. While principles of EDAs have already been explored, this thesis extends the state of the art by utilizing concepts of the semantic web. These concepts form a semantic model to realize the configuration of services for analysis workflows and specify the coordination among visualizations. The semantic model establishes and enforces contracts between views that manage data dependencies and expectations on event communication. A *runtime system* will be responsible for configuring the CMV system for a workflow. Finally, all use cases will be presented introducing novel visualization and interaction techniques for analyzing spiking network simulations.

1.4 Outline

The thesis is structured as follows: Chapter 1 outlines the motivation, requirements, and contributions of this work, Chapter 2 presents related work in the field of information visualization, CMVs, and EDAs. Following this, principles of semantic modeling and reasoning are introduced and brought together to derive semantic-aware CMVs systems that meet the requirements from Section 1.2. This enables the development of use case driven visualization services, facilitating the exploration of neural activity data. Subsequently, Chapter 3 focuses on three uses cases that all specify a semantic-aware CMV system for their analysis workflow. Chapter 4 discusses the benefits and limitations of the suggested approach while finally, Chapter 5 presents a conclusion and outlook of possible future work.

1.5 Publications

Parts of the here presented research has been published in peer-reviewed articles and journals. This section lists these and describes contributions by the author and co-authors.

Nowke *et al.* [2013] introduces visualizations for neuroscientific data based on a computational neural model of the macaque visual cortex. These visualizations form views in Section 3.1. In contrast to the original paper, this thesis embeds these views as services for a CMV system and introduces additional views. The author of this thesis designed, developed, and implemented the method, evaluated and incorporated feedback on the method's design, and wrote the paper. Maximilian Schmidt, Sacha J. van Albada, and Jochen M. Eppler developed the neural network, provided the simulation datasets, guided on the conception of views and outlined what key metrics of the simulation data needs to be visualized. Rembrandt Bakker provided the polygonal surface reconstruction via voxelization of the brain areas from the Scalable Brain Atlas. All co-authors, in particular, Bernd Hentschel, Markus Diesmann, and Torsten Kuhlen provided guidance in writing the publication.

Nowke *et al.* [2015] describes an early architecture and core principles which have been extensively formalized in Chapter 2. The author of this thesis developed and implemented the architecture, migrated existing views based on previous work to the architecture, guided on extending the CMV system, and wrote the paper. Daniel Zielasko contributed the LFP-View and wrote its description in the article. Benjamin Weyers, Bernd Hentschel, and Daniel Zielasko revised the article and contributed to its ideas. Torsten Kuhlen revised the article and presented valuable input while Alexander Peyser contributed to the underlying spike streaming concept for NEST simulations.

Nowke *et al.* [2018] forms the foundation of a CMV system described in Section 3.3. This thesis embeds the use case in PSE as a CMV system. Sandra Diaz-Pier contributed equally to this paper. The author of this thesis developed the interactive steering tool, designed the data flow for steering the main structural plasticity parameters, incorporated the envisioned views into the CMV system architecture, wrote the corresponding content, and coordinated further writing of this publication. Sandra Diaz-Pier, Alexander Peyser, and Abigail Morrison defined the use cases while Sandra Diaz-Pier evaluated the results and compared the process of parameter navigation with and without the steering tool. Alexander Peyser contributed the high performance computing knowledge to port and

optimize the code for super-computing usage. Benjamin Weyers, Bernd Hentschel, and Torsten Kuhlen provided scientific guidance. In addition, Abigail Morrison provided neuroscientific guidance and assessed the usability of the tool for generalized use cases. The author of this thesis, Sandra Diaz-Pier, Abigail Morrison, and Alexander Peyser wrote the paper.

CHAPTER 2

SEMANTIC-AWARE COORDINATED MULTIPLE VIEWS

2.1 Motivation

This thesis presents a PSE which enables the visual assessment of neural activity data resulting from neural network simulations. A PSE is an expert system and computing environment used to investigate problems in a scientific domain. The investigation is supported by providing a closed loop between the user, a computational steering process, and the visualization of its results [Parker *et al.*, 1998]. This PSE will enable distributed visualizations which are semantically linked to create CMV systems. The PSE consists of views, a communication infrastructure, a semantic model, a semantic reasoner, and a *runtime system* which provides access to all its functionality as shown in Figure 2.1. The purpose of the PSE is to create a CMV system based on a workflow that uses a subset of its functionality to provide analysis capabilities. To create this PSE, the scientific workflow plays a fundamental role in the analysis process. In context of the PSE, a workflow defines services and describes how these are interlinked to create a CMV system providing analysis capabilities. This workflow is described as part of the semantic model which is stored in an ontology (see Figure 2.1). Based on this ontology, a semantic reasoner is used to infer how services can be used for a workflow to form a CMV system. A *runtime system* controls the semantic model and uses the communication infrastructure to interconnect services. Then, required services are started and connected to form a CMV system.

To realize a semantic-aware CMV system, six building blocks are required:

1. a domain description encoded in a semantic model.
2. a workflow described in this semantic model.
3. a semantic description of services available in the PSE.
4. a semantic reasoner.
5. a communication infrastructure for services to exchange data.

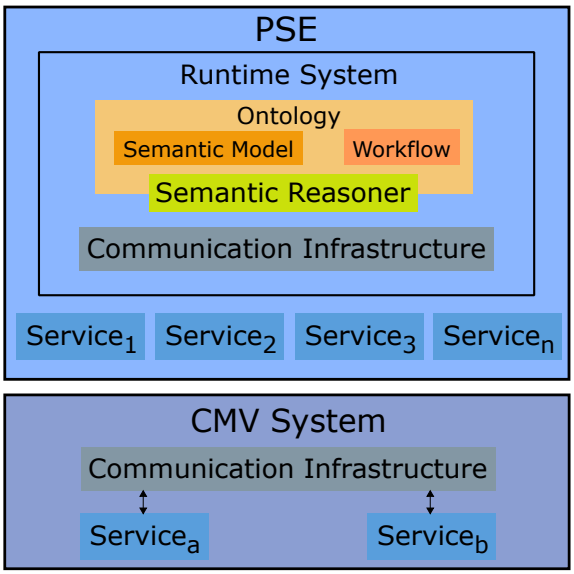


Figure 2.1: The PSE consists of all available services and the *runtime system*. The *runtime system* embeds a semantic model and describes workflows in an ontology. A communication infrastructure and a semantic reasoner are used to select and link services to create a CMV system for a specified workflow.

6. a *runtime system* that uses the semantic model, utilizes a semantic reasoner, control services, and allows to specify workflows.

To understand these building blocks for the development of a PSE, the following sections will present related work and outline the construction of semantic-aware CMV systems for the exploration of neural activity data. To this end, an introduction of information visualization, concepts of CMV systems, and EDAs are presented, followed by concepts of semantic modeling which will be used on top of an EDA.

To summarize, semantic-aware CMVs are realized as distributed loosely coupled applications termed *services*. To achieve loosely coupling and to form a PSE, services make use of a EDA. The resulting PSE enables the construction of a specific CMV system for a given workflow, and offers flexibility in adding or removing services to support a variety of workflows.

2.2 Related Work

In this section, I will first briefly discuss the field of information visualization which is the basis of interactive visualization. Next, I will outline CMVs which are a fundamental technique for relational data visualization. Subsequently, I will explain the design principles of EDAs.

2.2.1 Information Visualization

Information visualization is the art of transforming data into visual representations. This transformation leverages human perception abilities to extract insight from visual stimuli. Using computer systems, interactive information visualization allows for visualization

systems that enable interaction with the visual representations to explore the underlying transformed data with the aim to find insight.

According to Yi *et al.* [2007], visualization systems can be divided into three integral parts: the data component, the visual component, and the navigation component. For each component, user intents can be asserted: on the data component, a user intents to *select* data for investigation, then *filter* it according to a self-specified criteria, and finally to abstract data, meaning to aggregate and create derivations of new properties based on it. This idea follows the *information seeking manta* postulated by Shneiderman [1996]. To extract insight, the information seeking mantra states that users first intend get an overview of the data, then want to zoom and filter to finally request details on demand. For the visual component, as defined by Yi *et al.* [2007], user intents are summarized as *encoding*, *abstraction*, *elaboration*, and *reconfiguration*. Encoding characterizes the assignment of visual representations to data attributes. Abstraction and elaboration specifies to add or remove details in the visualization. Finally, a reconfiguration intent delineates possibilities to gain different perspectives on the data, e.g., by re-aligning plot axes in a time series. Users intents on the navigation component primarily concern the *exploration process*. This usually includes any form of interaction in space or time in the data to shift the focus of interest. The visualizations developed for the use cases presented in Chapter 3 follow this classification since each component can be identified.

Kalawsky [2009] emphasizes to incorporate the human cognition process into the design of interactive visualization methods. Following the conceptual characterization of Yi *et al.* [2007], CMVs construe a class of visualization systems that inherently adhere to this classification.

User interaction in information visualization systems can be summarized into five distinct subtasks according to Bowman *et al.* [2004]: *selection*, *navigation*, *manipulation*, *symbolic input*, and *system control*. One method of performing these subtasks in Virtual Environments (VEs) are *Extended PieMenus* [Gebhardt *et al.*, 2013]. Accordingly, interaction tasks will be utilized in several views in Chapter 3.

2.2.2 Coordinated Multiple Views

CMVs is a visualization technique applied in visualization systems that uses *two* or *more* distinct *views* to support the investigation of a *single* conceptual entity [North & Shneiderman, 1997; Roberts, 2007; Wang Baldonado *et al.*, 2000]. A conceptual entity is composed of a set of data and a specification to visually display it constitutes a view. CMV is a visualization technique intended to support exploratory data analysis [Roberts, 2007]. Roberts [2007] defines CMV systems as any instance where data is represented in multiple windows. This definition implies that contrasting representations are placed in subsequent views and that interactions within these views are *coordinated*. The term *coordination* refers to an interaction within a view which is linked in such a way that the same conceptual entity in all other views is affected. The overall idea is to offer interaction with a conceptual entity while emphasizing different details to understand the data.

Wang Baldonado *et al.* [2000] defines views in such a way that they allow users to explore a conceptual entity by presenting different aspects of information or by emphasizing its

differences. Moreover, views can differ in the visual representation of the same conceptual entity. Both definitions are non-orthogonal.

According to [Wang Baldonado *et al.*, 2000], CMV systems have generally three advantages compared to single view systems:

- they improve user task performance while exploring conceptual entities.
- they facilitate the discovery of unforeseen relationships in the data.
- they offer unification of the workspace, i.e., on one desktop.

In contrast, CMV systems also impose associated costs [Wang Baldonado *et al.*, 2000]:

- they increase cognitive overhead.
- they exhibit an increase in time and effort to learn the functionality of the system.
- they increase the load on the user's working memory.
- they introduce elevated cognitive effort for comparison tasks and context switches.

Nevertheless, CMV systems have successfully been used to uncover complex relationships, to relate datasets and scales, and to assist in context switches and comparative tasks [Wang Baldonado *et al.*, 2000]. To this end, CMV is an effective technique to extract insight from neuroscientific data.

To develop and leverage the potential of CMVs, Wang Baldonado *et al.* [2000] present common guidelines for designing CMV systems:

- **Rule of diversity:** CMVs are justified when there is a diversity of attributes, models, user profiles, levels of abstraction, or genres. In the later presented use cases, the available datasets consist of diverse attributes and levels of abstraction, indicating that the use of the CMV technique is justified.
- **Rule of complementarity:** the use of multiple views is amplified when they highlight correlations or disparities in the data. The later presented use cases aim at extracting insight about correlations and disparities in the data, suggesting that a CMV system can assist in finding these.
- **Rule of decomposition:** partition complex data into multiple views to create manageable chunks and to provide insight into the interaction among different dimensions. The visualizations from Section 3.1, Section 3.2, and Section 3.3 all partition the data in views, thus decomposing it into smaller manageable parts. To provide interaction among dimensions, views are linked.
- **Rule of space and time resource optimization:** balance the spatial and temporal costs of presenting multiple views with the spatial and temporal benefits of using them. The PSE allows to select and use a subset of available views, deriving a custom CMV system based on a workflow, thus enabling to concentrate on the currently relevant analysis task.
- **Rule of self-evidence:** use perceptual cues to make relationships among multiple views more apparent to users. Perceptual clues are realized by a *global distributed coordination space* which allows for linking conceptual entities throughout

the CMV system. For example, perceptual clues are given in the form of highlighting selections across multiple views displaying the same conceptual entity.

- **Rule of consistency:** UIs of multiple views should be consistent. While UIs are implementation details, the provided views adhere to the same interaction principles as far as the use the same display system. Consistent user interaction between traditional UIs and Virtual Reality (VR)-based interaction is still an open research question. However, views could potentially be expanded by a semantic description of their interaction concepts. This would make it possible to infer which views are using consistent interaction metaphors, therefore assisting in finding suitable coordinations.
- **Rule of attention management:** ensure that user attention is in the right place at the right time. While attention management is highly important, it is highly use case related. Techniques to detect if a view is affected by coordination are possible, but not in the focus of this work.

While these guidelines are useful considerations when developing CMV systems, these class of systems accompany a substantial impact on system requirements and developing costs: firstly, they introduce additional computational costs for rendering multiple views and synchronizing coordination. Secondly, an increase in display-real-estate costs is present to display views concurrently. Finally, increased costs in resources to design, to implement, and to maintain these systems are required due to their larger complexity.

To this end, the presented PSE will mitigate the complexity in the development and maintenance of views by utilizing fine-granular services that exclusively communicate via events.

Conceptual Distinctions and Interactions in CMVs

Wang Baldonado *et al.* [2000] describe three conceptual distinctions on views in a CMV system: the *selection* of views, their *presentation*, and *interaction*. The selection process concerns the identification of coordinated views for an analysis task. Their presentation can either be a sequential or simultaneous. Wang Baldonado *et al.* [2000] point out, when presenting views sequentially, many possible configurations of views may exist. However, the main benefit of CMV systems stems from interaction, in particular coordination. Here, Wang Baldonado *et al.* [2000] distinguish three basic coordination techniques: firstly, *navigational slaving* where navigation and selections in one view are simply propagated to all other views. Secondly, *linking* which connects a conceptual entity in one view to all occurrences in other views [Velleman & Velleman, 1988]. For instance, if a conceptual entity is selected in one view, all other views displaying the same conceptual entity select or highlight it. Both, navigational slaving and linking require a *coupling function* that relates entities to their counterparts in other views. Navigational slaving and linking are techniques mainly used in the views from Chapter 3. Lastly, *brushing* is a special form of linking, where a user highlights attributes of a conceptual entity and corresponding attributes of that entity are highlighted in all other views [Becker & Cleveland, 1987a; Roberts, 2007]. Linking can also be spatial and temporal, e.g., views that show animated sequences [Buja *et al.*, 1991].

All coordination techniques require a mechanism to share interaction among views and

linking and brushing require a relation between conceptual entities. To this end, the PSE facilitates a *global distributed coordination space* for the propagation of interactions.

Implications of Utilizing CMV Techniques

Roberts [2007] iterates on six fundamental areas a CMV system has to cope with and present universal implications designers have to face:

- **Data processing and preparation:** before any visualization, the underlying data needs to be preprocessed, cleaned, fused and related to facilitate knowledge discovery. The proposed PSE allows for decoupled data processing and preparation by services accessible to views. This enables the implementation of efficient data processing techniques without effecting visualization services.
- **View generation and multiple views:** view generation describes the conception of visualizations. This includes how to visualize data by selecting appropriate visualization techniques and how to interact with it. The suggested PSE offers flexibility to integrate novel visualizations by allowing to add or remove views and share interaction. This enables rapid prototyping, the identification of appropriate visualizations and interaction techniques.
- **Exploration techniques:** a main benefit of CMV systems is the ability to use a visualization technique most suited to the data and to link it to views focusing on other attributes; possibly utilizing a different visualization technique. This allows to apply exploration techniques like direct [Shneiderman, 1994] or indirect manipulation [Becker & Cleveland, 1987b]. In Chapter 3, examples of direct manipulation are given for exploration tasks in several neuroscientific use cases.
- **Infrastructure and tools:** infrastructure refers to mechanisms and models to realize coordination among views. Prominent examples are the constraint system by McDonald *et al.* [1990], a data centric approach inspired by relational databases [North & Shneiderman, 2000], or the coordination model by Boukhelifa & Rodgers [2003] which mimics the model view controller pattern [Krasner & Pope, 1988]. The coordination principles used in this thesis resemble Boukhelifa & Rodgers [2003] the most. An in-depth discussion is presented in the next section. As tools, Roberts [2007] understands either exiting CMV systems that offer customization features such as [North & Shneiderman, 2000; Weaver, 2004] or technology easing the development of CMV systems.
- **UIs:** a CMV system presents multiple views, consuming more display-real-estate in comparison to single view systems. Also, it offers interaction possibilities that put additional strain on users' mental load. To alleviate these effects, window- and session management are commonly applied in CMV systems. While the CMV systems derived here do not use window- or session management, both techniques reflect important additions to the PSE as future work.
- **Usability and perception:** both aspects concern the effectiveness of CMV systems, especially in comparison to single view systems. For instance, Ryu *et al.* [2003] explore cognitive strategies in CMV systems and suggest that users benefit from CMV systems if diverse attributes in the data are present, confirming the rule of diversity [Wang Baldonado *et al.*, 2000]. In addition, users benefit from orthogonal

combinations of views because cognitive interference may occur if similar views are presented simultaneously. However, Card *et al.* [1983] note that if two events occur within 100 ms, users may perceive these as causally related, potentially leading to misinterpretations. While usability and perception are undoubtedly important issues for designing CMV systems, no studies on these effects have been performed in this thesis.

In the following section, examples of CMV systems will be given.

Examples of CMV Systems

Selected examples of CMV systems can be found in [Keefe *et al.*, 2009; Matkovic *et al.*, 2008a; North & Shneiderman, 2000; Weaver, 2004]. Weaver [2004] presents *Improvise*, a system which allows to build CMVs interactively by the means of a shared-object model for coordination. Improvise provides an expression-based visual abstraction language that enables the definition of relationships in datasets and a fine-grained control over coordination. The visual abstraction language allows for flexibility in defining coordination by the means of scripting. This is realized by *live properties* which use a symmetric update and notification mechanism to coordinate views. Live properties model a data-flow that is used for coordination. However, it lacks a method to identify loops build by users that break the system [Weaver, 2004]. The mechanism of live properties is similar to the *global distributed coordination space* where services share coordination events: whenever a conceptual entity is changed, linked services will be notified and can react to it. Moreover, the PSE allows users to dynamically define participation in coordination while the CMV system is running.

North & Shneiderman [2000] present *Snaptogether*, a tool which provides a UI for specifying a formal description of related datasets. This formal description empowers users to define views without programming knowledge. Moreover, Snaptogether provides an application programming interface to develop additional views. Thus, Snaptogether is an example of one of Roberts [2007] fundamental areas of tools, geared to create CMV systems. North & Shneiderman [2000] note, similar to Wang Baldonado *et al.* [2000], that it is difficult to anticipate the need of users on how to link views, resulting in endless possible combinations. In this thesis, defining coordinations is part of the CMV system creation from within the PSE. While views encode the relationships to conceptual entities as part of their service implementation, participation in the *global distributed coordination space* is user-controlled through either using a Graphical User Interface (GUI) or by the means of scripting.

Boukhelifa & Rodgers [2003] describe a formal model for expressing coordination in *CViews*. The generic model is designed without any bias toward requirements on data, navigation concepts, or view synchronization paradigms. Its purpose is the synchronization of views, the tracking of visual and non-visual information, the avoidance of view explosion, and easing the multitasking demand for users. It is based on a data-flow model to share view and visualization configurations and focuses on three aspects:

- the representation of a conceptual entity in one view is not necessarily the same in another one. This makes the definition of how a manipulation of one conceptual entity relates to other views difficult.

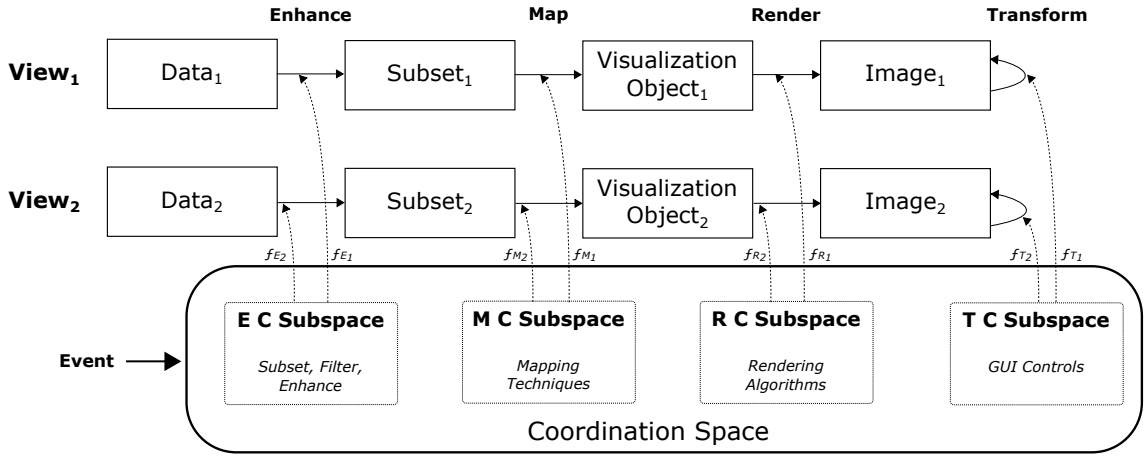


Figure 2.2: Events modify coordination objects within the coordination space. This triggers a notification to views which are registered via coordination objects. The coordination space is populated by coordination objects which link parameters to views and specify a translation function f for each subspace. The figure is modified from [Boukhelifa & Rodgers, 2003].

- UIs and conceptual entities in views need a specification of how and where-from they receive coordination.
- a CMV system should indicate which views and conceptual entities are coordinated.

As can be seen in Figure 2.2, this model spans a *coordination space*, subdivided into four subspaces in which *coordination objects* are contained. A coordination object links views and communicates coordination (shown as dashed-lines in Figure 2.2). Each coordination object contains a *translation function* f. This function relates a view parameter, expressed as the argument to f in Figure 2.2, for each subspace to a conceptual entity in a view. Views are then register to coordination objects in order to receive notifications when parameters are changed. A change in a coordination object, propagated via an event, is triggered through UIs. In comparison, the coordination model used by the semantic-aware CMV systems is conceptually similar. However, I slightly diverge from the definition of a coordination object and its responsibilities. Also, the coordination space, termed *global distributed coordination space*, is accessible to remotely connected views, e.g., a visualization concurrently utilizing different display systems. The subspaces spanned in the coordination space from Boukhelifa & Rodgers [2003] are modeled via concepts in an ontology and are attached to *coordination endpoints* provided by views. In addition, the translation functions are directly defined in views. A detailed description of the coordination model utilized in this thesis will be presented in Section 2.3.7.

CMVs and Immersive Visualization

HVEs are a category of display systems that use a VE in conjunction with traditional display devices like a monitor, tablet, or PowerWall to create a holistic user experience. The term hybrid refers to the combination of immersive and non-immersive environments used to display and interact with content. While VEs generally provide a more direct access to the spatial data encountered in scientific visualization, non-immersive environments are often better suited for traditional information visualization tasks, e.g., displaying and

interacting with a function plot, a chart, or a diagram. VEs can enhance a visualization tasks through immersive and intuitive UIs [Laha *et al.*, 2012; Raja *et al.*, 2004; Ware & Franck, 1994].

To leverage the benefits of immersive visualization, one contribution of this thesis is to enable seamless integration of views into visualization systems that utilize a wide variety of display devices concurrently, effectively permitting the design of HVEs. To this end, the featured PSE provides the means to distribute data, interaction, and coordination to display systems interconnected via a network. However, the diversity of the display-, input-, and output devices must be managed in such a way that the developed visualization system has minimal dependencies on certain device setups [Badam *et al.*, 2014], as for example realized in the ViSTA software stack [Assenmacher & Kuhlen, 2008].

The diversity of display systems and interaction devices necessitates special interaction techniques to provide a seamless user experience. To this end, Wang & Lindeman [2014] present an interaction technique that combines a VE, displayed via a head mounted display and a tablet, to provide efficient interaction with a scenery showing diversities in scale, perspective and dimension. Adding to this, Wang & Lindeman [2015] further explore diverse 3D interaction tasks, including object manipulation from different scales and reference frames. Of particular interest are the coordination mechanism which reduce transition gaps across 3D UIs and tasks. Hence, the authors propose four coordination mechanisms for seamless interaction between devices which are termed interaction contexts. An evaluation of one the interaction contexts in a user study suggests that despite the complexity in handling context switches, users tend to perform well.

While a CMV system was developed as part of a proof of concept for a use cases presented in Chapter 3, more rigorous exploration of CMV systems utilizing HVEs is required as potential future work.

2.2.3 Event-Driven Architectures

An EDA is a software design principle based on the generation, detection and consumption of *events* among loosely coupled, distributed software components which are termed *services* [McGovern *et al.*, 2006; Newman, 2015]. Hence, an EDA is characterized as a collection of distributed programs with defined functionality and interfaces that exchange information via *events*. The paths events are distributed is termed an *event-flow*. This concept is very similar to the fundamental working principles of a brain where a neuron (service) exchanges information (events) via synapse (event-flow). Synapse are dynamic; they can grow or recede from neurons; analogously, depending on a workflow, the event-flow among services can change.

One benefit of EDAs is the loosely coupling among services which enables software systems to be flexible, scalable, and highly integrative. Flexibility allows for non-complex service implementations while higher functionality is realized by sticking services together. Scalability is achieved by distributing services to multiple computing resources. Integration enables applications, developed with different software frameworks such as ViSTA [Assenmacher & Kuhlen, 2008] and NEST [Gewaltig & Diesmann, 2007], or other programming languages, to be used within an ecosystem. This is possible because applications retain independence by only sharing data via events over application boundaries. In context of data visualization, utilizing EDAs allows for using various display

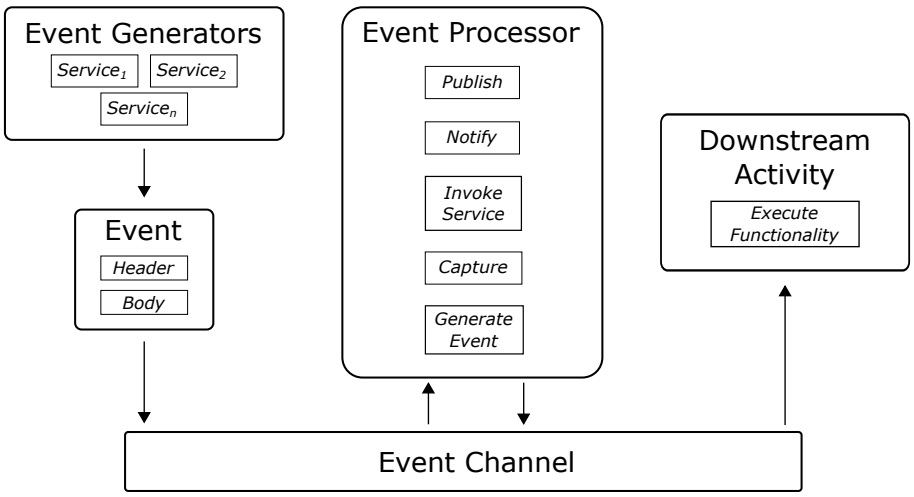


Figure 2.3: Event-flow layers in an EDA: event generators create and emit events. An event contains a header specifying its designator and a body to hold state information. Event processors receive and process events over an event channel. Eventually, events trigger the execution of functionality as part of a downstream activity. The figure is simplified and adapted from [Michelson, 2006].

systems and therefore the development of HVEs.

As services in an EDA communicate via events, they are characterized by the ability to *emit*, *detect*, and *react* to events. Event transport is realized by a *communication infrastructure*. A communication infrastructure delivers events over a network or other means of inter-process communication, e.g., MPI[1], RPC[2], or CORBA[3]. However, EDAs emphasize loosely coupling which operates under the assumption that services only react to events, without knowledge of where events are originating from. Thus, EDAs are push-driven, do not enforce handshakes between services, and share data on a *fire and forget* principle. Reaction to an event is part of a logical processing layer which characterizes an event as a change of state that merits attention from the system.

Michelson [2006] distinguishes four *event-flow layers* as depicted in Figure 2.3. Any event-flow starts with an *event generator* and may culminate with the execution of *downstream activities*. To this end, the first layer consists of event generators that create and emit events. An event generator only transpires events and has no knowledge of any subsequent processing. Events from an event generator can optionally be processed by an event preprocessor for filtering and routing purposes before being passed to the second layer. The second layer, termed *event channel*, uses a communication infrastructure to pass events to all subsequent layers. An event consists of a header and a body [Michelson, 2006]. The header contains a designator that holds information about the event's content, e.g., its name, a time stamp, or the event generator it originates from. The event header can be used by an event preprocessor to detect and route events to a designated *event processor*. The event body encodes a state change. An event must encode the entire state change because event processors cannot actively request additional states from the event's originator. By design, event generators and processors use *stateless commu-*

[1] `http://mpi-forum.org`
[2] `https://tools.ietf.org/html/rfc5531`
[3] `http://www.omg.org/spec/CORBA/`

nication. The third layer contains all event processors. As shown in Figure 2.3, an *event processor* observes events on the event channel, evaluates these against event processing rules, and takes action according to its specification. Event processors might process events independently or in a context sensitive manner. Hence, an event processor has the following responsibilities when reacting:

- publish events to trigger downstream activities.
- notify system components.
- invoke services.
- capture events for recording.
- generate and emit events to the event channel.

The execution of *downstream activity* is the fourth and final layer and represents the direct response from the EDA while not triggering any additional logical event processing.

Michelson [2006] suggests that EDAs are best used for asynchronous flows of information and presents three possible *event processing* styles:

- **Simple processing:** when an event is detected, only downstream activities are triggered.
- **Stream event processing:** an event processors monitors and filters events to only trigger specific downstream activities.
- **Complex event processing:** a confluence of events is monitored for a time period, containing events of different types, to determine event correlations. Event correlation may be causal, temporal or spatial. This processing technique requires an *event interpreter* which performs pattern recognition or other forms of correlation detection techniques. Subsequently, new events are emitted to be processed by the EDA.

In summary, EDAs are a software architecture design principle that enable extensibility, scalability, and flexibility by decoupling complex system components into smaller executional parts that communicate via events. Besides flexibility and extensibility, EDAs scale well in comparison to monolithic software applications and can exploit heterogeneous computing platforms. However, the benefits also introduce complexity to the communication infrastructure, the invocation of services, event processing, and the composition of services to form higher functionality out of the smaller executional parts. In the following sections, I will present and apply techniques of semantic reasoning to mitigate some the costs of utilizing EDAs.

2.3 Semantic-Aware Event-Driven Architecture

To develop a PSE that realizes the requirements outlined in Section 1.2 to create CMV systems suited for various visual analysis workflows and a shifting focus in a scientific analysis process, an EDA will be employed. Using an EDA allows for covering diverse scientific workflows in the analysis tasks, the development and integration of additional views, and reusing existing ones. Due to its flexibility, it enables the integration of heterogeneous data and concurrently running interactive immersive visualizations, coexisting on

multiple systems. In principle, whenever a new visual analysis task arises, existing applicable views should be reusable. The creation of additional views should only be required if there are more appropriate visualization or interaction techniques. Moreover, existing views, specialized to an analysis task, should coexist with new views without requiring modification which facilitates a transition back to a previous analysis workflow.

To this end, this section focuses on three fundamental parts to realize the PSE as a *semantic-aware* EDA:

1. the introduction and definition of the PSE components along with their conceptual mapping to an EDA.
2. a communication infrastructure, serving as event channel, that enables inter-process communication for services.
3. a semantic model that describes visual analysis tasks as workflows, enables user to create a CMV system via the invocation of services and definition of view coordination.

Implementing the PSE as an EDA enables flexibility, extensibility, scalability, the definition of services, and the creation of views for CMV systems. However, utilizing an EDA and its loosely coupling results in managing services for visual analysis tasks. To this end, a semantic model in conjunction with semantic reasoning [Decker *et al.*, 2000] is used to assist and counteract the increased complexity in defining workflows, managing services, establishing event communication, and guarantee valid event exchange based on the requirements of services.

A communication infrastructure enables services to communicate via an event channel based on the publish and subscribe paradigm. The overall system's complexity can then be divided into smaller parts, actualized by services realizing views. This enables the definition of services and interfaces with clear functional responsibilities increasing the system's flexibility and maintainability. The communication infrastructure is also used to realize the coordination of views. Since view coordination is transported over an event channel, it allows views to utilize a variety of display systems most suitable to an visual analysis task. Moreover, it allows services to be developed in different programming languages. This closes the gap between the requirements of highly interactive visualization systems to leverage more hardware-oriented programming languages such as C++ to achieve interactivity and community tools in neuroscience which usually utilize Python. One advantage over existing CMV systems is the *dynamic control* over the coordination of views which can be changed by the user at runtime.

A semantic model, used by a *runtime system* and stored in an ontology, describes workflows as set of services required for this task, how they can be invoked, and linked. The *runtime system* provides a UI for the configuration of the PSE and enables to create and restore CMV systems for workflows. Basically, a workflow describing an analysis task enumerates on which services are required and how these are interconnected. As services communicate via events, this interconnection is termed an *event-flow network*.

In the following sections, I will present a definition of services, outline the communication infrastructure, the semantic model, define the event-flow network, and introduce the *runtime system* that make up the semantic-aware EDA. In addition, I will present a formalization of the system to clarify each part's function. Lastly, I will explain the coor-

dination model realized within the proposed PSE and provide an example on how it can be extended with new services.

2.3.1 Services, Slots, and Events

Services are the central part of an EDA to realize a functional system. In particular, any CMV system created within the PSE for an analysis task consists of a set of invoked services which are interconnected. Thus, a *view* in a CMV system is a service that deals with any kind of conceptual entity to be displayed.

A *service*, as part of this thesis, is defined by the following properties:

- a service is an executable program.
- a service must be invoked to be used within the system.
- a service offers functionality by the means of *slots* which define its interface.
- a service communicates exclusively via events.
- a service is allowed to run concurrently by invoking it multiple times.

For instance, a service can be a view displaying a visualization, or a data provider that loads data from a source and publishes the data via events.

In contrast to this definition, the World Wide Web Consortium (W3C) defines a web service as a software system designed to support inter-operable machine-to-machine interaction over a network. It has an interface described in a machine-processable format such as Web Services Description Language (WSDL)[4] Other systems interact with a web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards[5] Notable differences between web services and services as defined here are a binary serialization format to achieve performance suited for real-time visualization systems, a publish and subscribe communication paradigm, and an interface description based on a semantic model which allows semantic reasoning to be used in comparison to WSDL.

Issarny *et al.* [1998] place the following demands on services:

- the functionality offered by a service must match the one expected by a connecting service.
- the connectors used for handling interactions among services must provide communication protocols that conform those expected by services.
- the overall architecture must provide the desired quality in terms of criteria as diverse as dependability, efficiency, scalability, security, timeliness, and usability.

The first demand is met by modeling functionality of services and its expectations on event communication in an ontology as outlined in Section 2.3.3. The second demand is met by guaranteeing that functionality, i.e., downstream activity, of services is only accessible via service slots. Slots are tied to specific events and slot connections must have the same semantic description, hence, slots of different events cannot connect. The *runtime system*

[4] `https://www.w3.org/TR/wsdl`
[5] `https://www.w3.org/TR/ws-gloss`

will guarantee this property as part of the event-flow network construction, as outlined in Section 2.3.5. The last demand is partly met by the conceptual idioms of utilizing EDAs, e.g., scalability, while the rest are requirements on service implementation details.

To interface functionality of services, *slots* are used. Slots are the only means to communicate with other services. To this end, a slot is strongly tied to an event type and can either send or receive events of this type. It acts as an unidirectional start- or endpoint for event communication. Hence, services use slots to send or receive events over an event channel. A slot is uniquely identifiable throughout all running services within the PSE. Event communication requires two distinct type of slots:

- a *subscribing slot*, also termed *in-slot*, only receives events.
- a *publishing slot*, also termed *out-slot*, only emit events.

A subscribing slot can connect to multiple publishing ones, as long they send and receive the same event type and have the same semantic description. However, events received by subscribing slots cannot be traced to their point of origin. A publishing slot is conceptually equivalent to an event generator; similarly, a set of subscribing slots, associated with service logic triggering downstream activity within a service, is equal to an event processor as defined by Michelson [2006].

Finally, an event is defined as follows and equivalent to [Michelson, 2006] definition:

- an event is the only means to transport data among services.
- an event is transported over an event channel.
- an event is encapsulated in a message, consisting of an event header and body.

Events used within the PSE can be mainly categorized into *data events* and *coordination events*. Data events are produced and emitted from *data provider services* to transfer data to services operating on these. Coordination events are emitted from views to share, e.g., selections of conceptual entities, or camera positioning to realize navigational slaving.

Figure 2.4 illustrates the relationship between services, events, slot types, the event channel, and downstream activities. In this example, $Service_1$ emits an event consisting of a header and a body via an event channel to $Service_2$ and $Service_3$, because in-slots of $Service_2$ and $Service_3$ are both connected to the $Service_1's$ out-slot. In addition, $Service_2$ emits another event to $Service_3$. Whenever an event is received by an in-slot, corresponding downstream activity is triggered.

With this terminology, formalization of services, slots, and downstream activity is possible and will allow to combine an EDA with a semantic model later on:

Definition 2.3.1. Let an EDA E be defined as a 5-tuple $E = (S, \Omega, \Omega^*, \omega, \tau)$ such that S is a finite set of *services*, Ω is a finite set of *operations* (equivalent to downstream activities), and Ω^* is a finite set of instances of operations called *slots*. A function ω maps each service $s \in S$ to a subset of operations $\Omega_s \subseteq \Omega$ such that:

$$\omega : S \ni s \rightarrow \Omega_s \subseteq \Omega$$

A function τ maps each operation $o \in \Omega$ to a subset if slots (also denoted as instances of this operation o) $\Omega_o^* \subseteq \Omega^*$ such that:

$$\tau : \Omega \ni o \rightarrow \Omega_o^* \subseteq \Omega^*$$

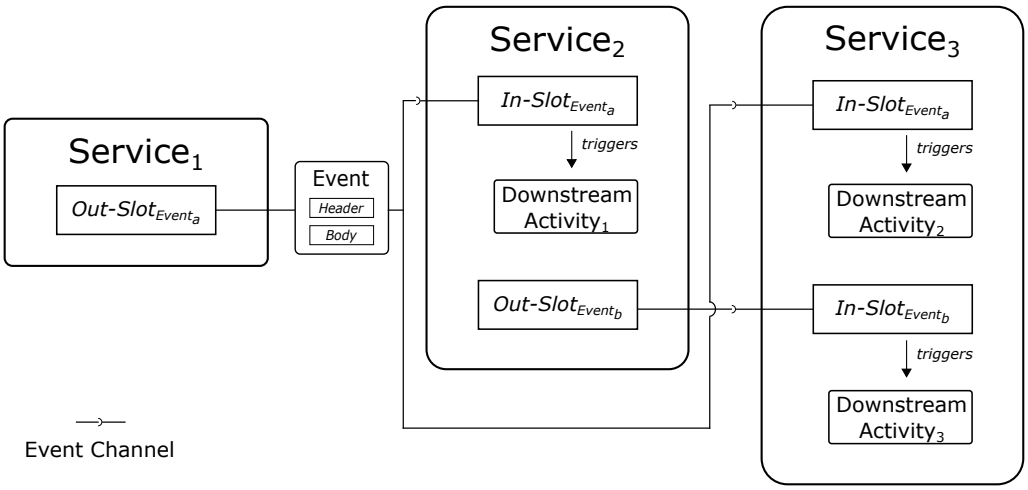


Figure 2.4: An event channel transports events among services. Here, $Service_1$ emits an event to $Service_2$ and $Service_3$ via its out-slot. The in-slots of $Service_2$ and $Service_3$ receive $Service_1's$ event and trigger downstream activity. Additionally, $Service_3$ receives an event from $Service_2$.

The next section will introduce the communication infrastructure that provides the event channel for event communication.

2.3.2 Communication Infrastructure

To share events among services, a communication infrastructure is required to realize an event channel for an EDA. To this end, the *nett* messaging framework was developed as part of this thesis. The communication infrastructure consists of three central components as depicted in Figure 2.5:

1. the *nett* library which provides event-serialization through schema definitions and network transport via slots.
2. the *nett-python* module which is build on top of nett and exposes the nett library's functionality to Python.
3. the *nett-semantic* module which augments the nett library's functionality with semantic descriptions about slots, grants control over these to connect or disconnect, and realizes their central announcement.

In the following, each component will be presented in more detail.

nett

The nett library is an open source C++ network library[6] to exchange messages across applications. The library's design is based on the publish and subscribe communication idiom with the goal to enable flexible and loosely coupled systems. One of the library's design intentions is the automatic generation of serialization code for user-defined events

[6] `https://devhub.vr.rwth-aachen.de/VR-Group/nett.git`

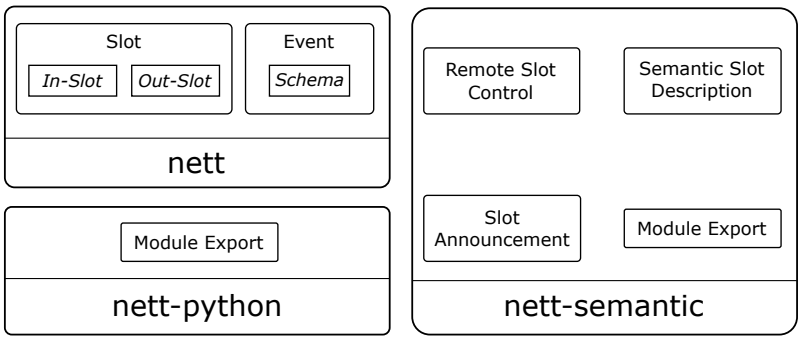


Figure 2.5: The communication infrastructure consists of three parts: the first part, termed *nett*, provides slots for sending and receiving events. Each event is defined and generated via a schema. The second part, called *nett-python*, exposes the nett library's functionality to Python. The final part, *nett-semantic*, augments *nett* with semantic descriptions for slots, provides control capabilities, and is responsible for the announcement of service slots. In addition, this functionality is exposed via a module to Python.

and compatibility with various programming languages. To this end, the nett library abstracts low-level network programming in a cross-platform library based on ZeroMQ[7] and Google's Protocol Buffers[8]. While ZeroMQ provides basic network communication, Protocol Buffers adds highly flexible serialization capabilities with the focus to enable distributed event-driven systems. To combine both feature sets, nett offers *slots* to service developers as a means to transfer events among services. Slots exist in two flavors: firstly, publishing slots, also termed *out-slots*, sending events. Secondly, subscribing slots, also termed *in-slots*, which receive events from possibly multiple publishing slots.

To use slots, a service developer has to initialize the nett library with an endpoint, i.e., an IP address. Subsequently, this allows to create either in-slots or out-slots. Whenever a publishing slot is created, a *slot tag* consisting of an arbitrary string must be specified to indicate the event's designator. A subscribing slot can then connect to several publishing slots as long as the event designator match. An event is comprised of the event's data and a header as introduced in Section 2.2.3. The header contains a routing designator for its delivery termed a *slot tag*. The event's data encodes the actual data to transport. To define an event, an event schema is used which describes the event's data as presented in Listing 2.1.

Listing 2.1: An exemplary event definition schema: a map is defined with a string and a vector of floating point values as a key-value-pair. Nested compositions of event definitions are allowed.

```
message float_vector_message
{
  repeated float value = 1;
}

message map_string_float_vector_message
{
  string name = 1;
```

[7] http://www.zeromq.org
[8] https://developers.google.com/protocol-buffers

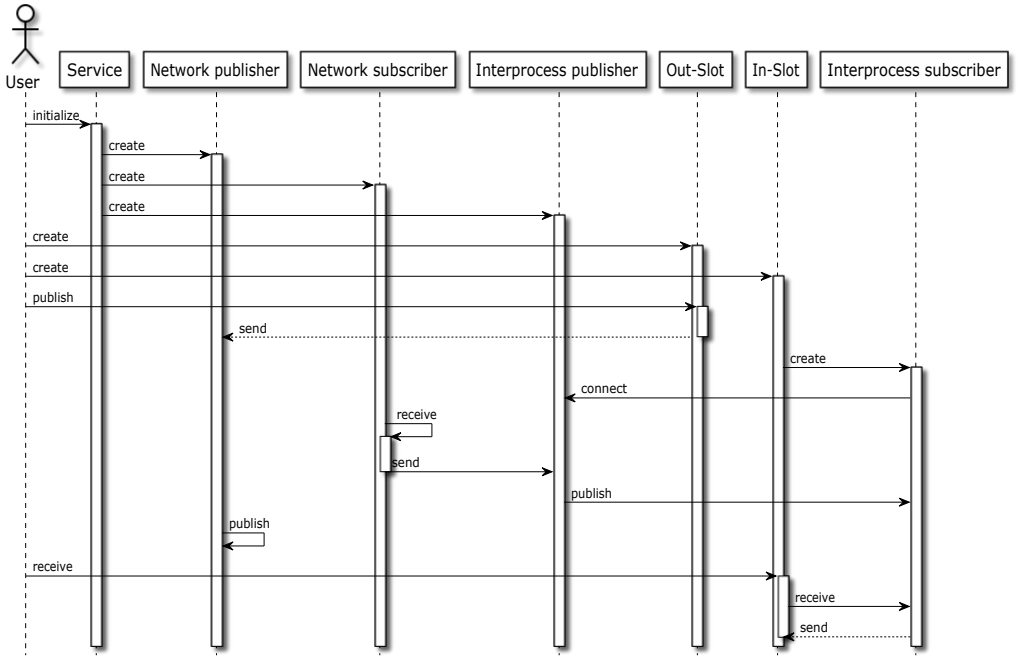


Figure 2.6: A nett service only exposes *one* network publisher and network subscriber allowing multiple slots to use one IP address and port. The delivery of events to their designated in-slots is performed using a shared memory communication channel via an inter-process publisher and subscriber. Events published by a service via an out-slot are forwarded to the network publisher while all event received by a service first arrive at the network subscriber and are then forwarded to their respective in-slot.

```
    map<string, float_vector_message> value = 2;
}
```

Once an event schema is provided, the event's serialization code stub is created for a target programming language and can subsequently be used within services to create slots using the event. This allows event definitions to be shared among services, ensuring binary compatibility and effectively building up a shared vocabulary. To finally send an event, a publishing slot typed to the provided event definition is created. Consecutively, an event is populated with data and published via the publishing slot. To receive the event, a subscribing slot typed to the same event definition is required. Receiving an event is a non-asynchronous call and access to the event's data can only conclude once this call has returned.

One of the library's design goals is that a service only occupies one endpoint to send and receive events necessitating *slot tags*. This has the advantage that the number of required communication endpoints only scale with the number of services invoked, regardless of the number of publishing slots offered by a service. To this end, the nett library ensures that a service only creates one *network publisher* and one *network subscriber* visible on the network layer. Event communication is entirely handled by these. As illustrated in Figure 2.6, a service creates a network publisher, a network subscriber and an inter-process publisher. Whenever an out-slot is requested to publish an event, the out-slot forwards the event to the network publisher which pushes the event to the network. On the other hand, when an in-slot is created, an inter-process subscriber is created which

directly connects to the inter-process publisher. The inter-process subscriber will filter all events received via the inter-process publisher and forwards the event to its corresponding in-slot if its *slot tag* matches the event's slot tag encoded in the event header. When an event is received by a network subscriber, the event is unpacked and the slot tag examined. The event body is then passed to the matching in-slot. Inter-process communication is used in order to avoid additional network strain.

nett-python

The nett-python module is an open source library[9] exposing the functionality of the nett library such that services can be implemented directly using Python while still being compatible to services written in C++. Because the Python programming language is widely used in the field of neuroscience, nett-python enables neuroscientists to, e.g., control views via slots or extend the PSE with services to cover different workflows fostering re-usability and extensibility.

The module is developed in such a way that previously defined event schemata can be reused over language barriers. Thus, the nett-python module allows for rapid prototyping of views and UIs that benefit from the cross platform compatibility of Python. In addition, the magnitude of available software packages like the Elephant, a package for analysis of electrophysiology data[10], allows for the integration of existing community tools. Moreover, the module enables instrumentation of PyNEST [Eppler *et al.*, 2009] and CyNEST [Zaytsev & Morrison, 2014] simulation scripts for the neural simulator NEST as presented in Section 3.3, demonstrating an in situ visualization and steering CMV system based on nett-python.

nett-semantic

The *nett-semantic* library extends the nett library's functionality with three essential features to enable a semantic-aware PSE and exposes its features to Python as a module:

- a slot announcement mechanism for services.
- remote slot controlling for services.
- slot augmentation by a semantic description.

Slot announcement enables the management of slots for CMV systems by publishing each service's slot to the central management instance called *runtime system*. Whenever slots are created, an announcement is send to the *runtime system*. Vice versa, when slots are destroyed due to a service shutdown, the *runtime system* is notified. This allows for controlling a slot by issuing a command in the *runtime system* to connect or disconnect from out-slots. Also, this enables *dynamic coordination* in a CMV system at runtime since coordination is realized via slot communication.

For slot announcement, nett-semantic adds a router pattern communication idiom on top of nett. The router pattern allows for multiple connections from services to the *runtime system* in order to announce slots. To manage slots, nett-semantics adds a communication

[9] `https://devhub.vr.rwth-aachen.de/VR-Group/nett-python.git`
[10] `http://neuralensemble.org/elephant/`

channel termed *control slot* to each service which handles connection and disconnection commands.

To semantically describe a slot, it is linked to a semantic model: a slot is required to specify a *semantic identifier* next to its slot tag. This semantic identifier is used to extend the semantic model when a slot announcement is received by the *runtime system* (see Section 2.3.3). The semantic identifier in conjunction with its used endpoint and slot tag identifies each slot uniquely in a CMV system.

Finally, nett-semantic exposes all of its functionality in a Python module. This allows scripting the construction of a CMV system and its specific slot connections.

2.3.3 Semantic Modeling

One of the central contribution of this thesis is the use of semantic modeling and reasoning to construct workflow-sensitive CMV systems. The semantic model shares a common, machine deducible understanding of services and their interconnections to reflect an analysis task by constructing a specialized CMV system fitting the workflow. It assists users in the selection, invocation, and the interlinking of services by the means of semantic reasoning. To utilize semantic reasoning, a knowledge base is required that encodes workflows, relates services to workflows, defines how services must be linked to reflect the workflow, and capture the current state of the system, e.g., what services are currently running, which slots they offer, and how they are connected.

Semantic modeling is the process of creating and organizing knowledge bases. A semantic model, encoded in an ontology, is a machine-readable knowledge base from which relationships of described things can be inferred. Its primary purpose is the classification of these things in terms of their semantic meaning. Ontologies are commonly used to formalize, share, and reuse knowledge bases. An ontology is formed from a set of *concepts* and *relations* between concepts. In addition, *instances*, also called *individuals*, describe a particular occurrence of a concept. A concept resembles a thing, an idea or a subcategory of a domain and is described by properties through an attribute-value mechanism [Decker *et al.*, 2000; Wang *et al.*, 2004]. A property describes an attribute of a concept and imposed restrictions. Hence, an ontology is a formal and explicit domain description of concepts, relations to express hierarchies of concepts, and individuals.

On ontology can be formalized as follows:

Definition 2.3.2. An ontology is defined as a 5-tuple $O = (C, H^C, R, H^R, I)$ where C represents a finite set of concepts. H^C defines a subsumption hierarchy representing a hierarchy of concepts as a relation:

$$H^C \subseteq C \times C$$

$H^C(c_2, c_1)$ denotes that c_2 is a subconcept of c_1. R represents a set of relations that relate concepts to one another:

$$R \subseteq C \times C$$

Similar to H^C, H^R, denotes a hierarchy of relations:

$$H^R \subseteq R \times R$$

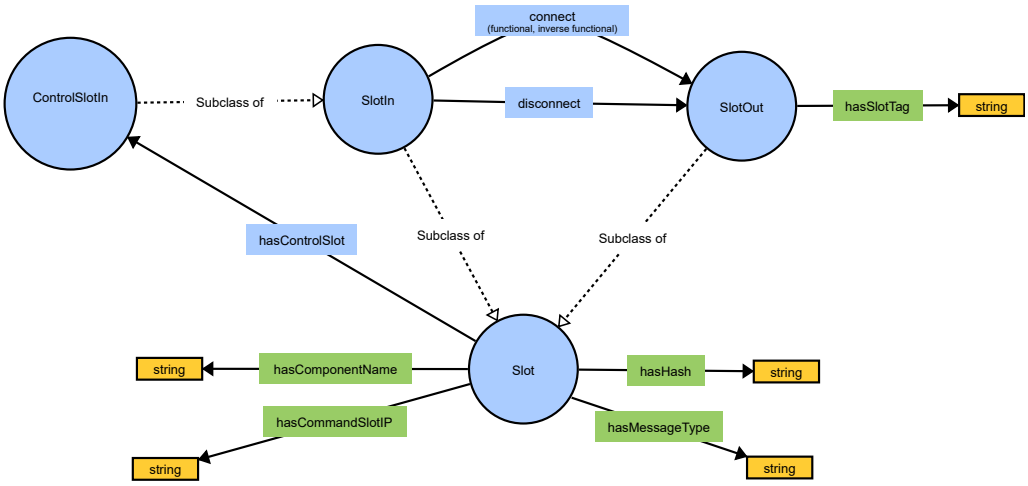


Figure 2.7: Graphical representation of parts of the semantic PSE model: concepts are depicted as blue circles. The *Slot* concept is parent to both subconcepts *SlotIn* and *SlotOut*. Green boxes represent data properties attached to concepts while blue boxes represent object properties relating concepts to one another.

Accordingly, $H^R(r_2, r_1)$ denotes that r_2 is a subrelation of r_1. Finally, I is a finite set of instances of concepts in C, assigned by an instance relation $R_i \in R$:

$$R_i \subseteq C \times C$$

An ontology is primarily build up of concepts and subconcepts. A concept defines a classification of individuals, i.e, a group that shares common characteristics. Hence, if an individual is a member of a concept it falls under the concept's semantic classification. Semantic reasoning operates on relations between concepts. To this end, an *object property* $\in R$ is a relation that links two concepts. For instance, an object property "connect" can relate the two concepts "SlotIn" and "SlotOut" as depicted in Figure 2.7. *Data type properties*, also known as attributes, relate instances of concepts to literal values as shown in Figure 2.7 as green boxes.

To model ontologies, the Web Ontology Language (OWL)[11] is used. OWL is a computational logic-based language designed to represent knowledge bases in the Semantic Web. Ontologies described in OWL can be referenced from other OWL ontologies allowing to separate and structure knowledge bases. This characteristic is utilized as described in Section 2.3.4. OWL is build on top of two components: the Resource Description Framework (RDF) and the Resource Description Framework Schema (RDFS) as illustrated in Figure 2.8. RDF is a W3C recommendation to standardize the description of resources. In context of OWL, RDF defines data types while RDFS models concepts using RDF. OWL then adds the description and definition capabilities to relate concepts. RDF is based on a labeled, directed graph. Its basic building block is a triple of "object-attribute-value", also expressed as $A(V, O)$ defining a relation that an object O has an attribute A associated with a value V. To allow for semantic reasoning, an OWL model is transformed into a description logic using triplets of the form "subject-predicate-object". Based on this description, *SPARQL*, a SQL-like query language, is used to inquire semantics and infer

[11] `https://www.w3.org/OWL/`

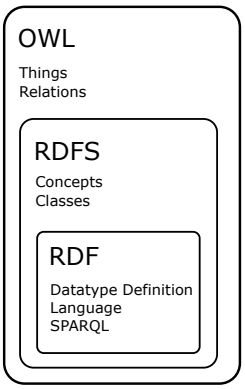


Figure 2.8: OWL is build of RDF and RDFS. RDF provides data type definitions and SPARQL, a query language for RDF graphs; RDFS defines concepts. OWL models relations.

logic. SPARQL contains capabilities for querying graph patterns along with their conjunctions and disjunctions[12]. In addition, it supports aggregation, subqueries, negation, expression creating values, extensible value testing, and can constrain queries by a source RDF graph. Evaluating a SPARQL query yields result sets or RDF graphs. For example, SPARQL allows to check whether an individual is an instance of a particular concept, to find individuals of a given concept, or how concepts and their individuals are related to each other. Concepts can also be matched against a set of object- or data properties. Listing 2.2 demonstrates a simple SPARQL query that retrieves all individuals matching the concept *C*.

Listing 2.2: Querying all individuals matching the concept *C*:

```
select ?s where { ?s rdf:type <C> }
```

Modeling an ontology consists of five elementary steps:

1. define concepts.
2. arrange these concepts in a taxonomic hierarchy.
3. describe properties, define allowed values for such, and attach them to concepts.
4. create instances of concepts and provide values for their properties.
5. iteratively refine the ontology.

Graphical tools such as protégé can assist in the modeling of ontologies. Protégé is a free, open source ontology editor developed at the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine[13]. Advantages of protégé are its W3C standard-conform output, its visual ontology creation, and refactoring capabilities.

Semantic modeling forms the basis for the PSE that utilizes a semantic architecture. To this end, the next section will introduce a semantic model in conjunction with an EDA.

[12] https://www.w3.org/TR/2013/REC-sparql11-query-20130321/
[13] https://protege.stanford.edu

2.3.4 Semantic-Aware Architecture

To realize a semantic-aware EDA, a semantic model is required which models services, slots, and describes their functionality by the means of concepts, individuals, relations and properties. This semantic model is based on two parts:

- a *static* knowledge base, termed *persistent ontology*.
- a *dynamic* knowledge base, termed *runtime ontology*.

The persistent ontology models the concept of services, their configurations, slots, and corresponding individuals. The persistent ontology encodes information that typically does not change while the PSE is in use. For example, configurations for services which specify how services can be invoked, usually are immutable and therefore stored in the persistent ontology. On the other hand, the runtime ontology keeps track of the current system state of the PSE, i.e., which CMV is currently used, and extends the persistent ontology by adding concrete individuals of concepts. Thus, depending on invoked services, available slots, and established connection among services, the runtime ontology changes. Both ontologies are managed by the *runtime system* which is a service monitoring the invocation of services and will be presented in detail in Section 2.3.6.

The overall idea of the semantic model is to capture the state of the PSE and assist in forming a CMV system depending on a workflow. To this end, services are associated with concepts. Whenever a service is invoked, an individual of its associated concept is created and added to the runtime ontology. Analogously, all slots of a service create individuals of their corresponding slot concept. This requires that services and their slots are augmented with a *semantic identifier*. The semantic identifier represent the associated concept to be instantiated and must match against a concept defined in the persistent ontology. The following definition formalizes this process:

Definition 2.3.3. Let $O = (C, H^C, R, H^R, I)$ be a ontology and Σ be an alphabet containing all alphanumeric characters. Let w denote an arbitrary semantic identifier:

$$\Sigma = \{a, ...z, A, ..., Z, 0, ..., 9\}$$

$$w \in \Sigma^* \mid w \neq \varepsilon$$

The semantic identifier w is an element over the Kleene closure of Σ but must not be the empty string ε. Let f be a function that accepts a $w \in \Sigma^*$ and a concept $c \in C$ and maps to an individual $i \in I$:

$$f : \Sigma^* \times C \to I$$

An operation S adds this individual $i \in I$ of f to the set I in O:

$$S := I \cup \{i\}$$

The function f will be evaluated for each service and its slots once invoked. Subsequently, the operator S will add the created individuals to the ontology O.

As illustrated in Figure 2.9, services and slots are augmented with semantic identifiers which are associated to concepts in the persistent ontology. Once a service is invoked, the *runtime system* is notified of the service and its slots. Subsequently, the *runtime system*

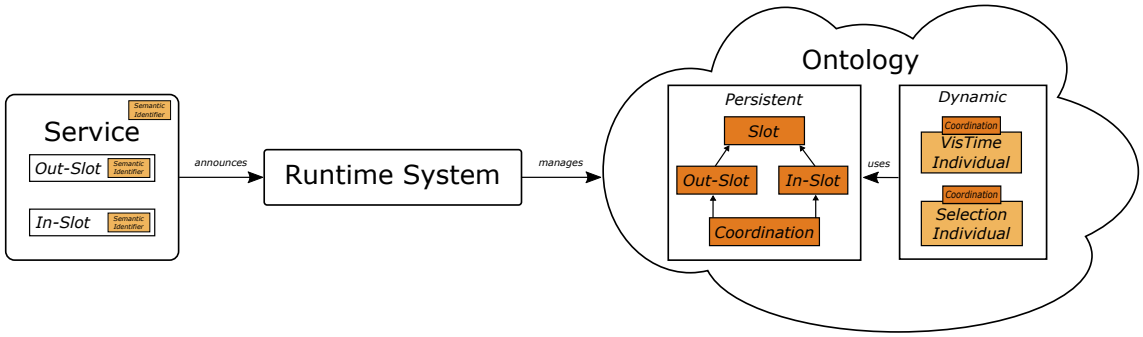


Figure 2.9: A service exposes slots to trigger downstream activity. The service and its slots are augmented with semantic identifiers. When a service is invoked and creates its slots, the semantic identifiers are announced to the *runtime system*. The *runtime system* then matches the semantic identifiers against concepts in the persistent ontology and creates corresponding individuals in the runtime ontology. Then, semantic reasoning can be used to create connections between in- and out-slots.

creates individuals of the matching concepts which are added to the runtime ontology. Analogously, whenever a service is shutdown, the corresponding individuals are wiped. Thus, the runtime ontology keeps track of the current system state, e.g., what services are currently invoked and which slots are available and connected.

To allow for semantic reasoning, the persistent- and runtime ontology are combined. As each ontology is encoded as a RDF graph, extending one ontology by another is performed by inserting the RDF triplets into one unifying graph representing the union of both ontologies. Following, the unified ontology is used to infer which slots can be connected to reflect an analysis task and hen creating a CMV system. To infer which services are needed, workflows for each analysis task are modeled in the persistent ontology. The invoked services and their formed connections is termed *event-flow network*. Next to the state management, the deduction of a valid event-flow network forming a CMV system is the primary task of the semantic model.

Augmenting the EDAs with semantic modeling allows to reduce event explosion within the PSE by linking slots to concepts while using the same *slot tag* enabling events to be used in multiple contexts. Event explosion refers to the effect that when the event type is used as a description of its content, new events are required whenever their content interpretation is changed even though the encoded data stays the same. For example, consider the case where a service in-slot expects a time series event, encoded as a vector of floating point values, to visualize the event's content in a plot. This implies that all services operating on a time series event interpret the event's content the same. But are the values of the time series encoded in seconds or milliseconds or any other unit? There is no way to systematically enforce the same interpretation of the time series data, except by convention, making it prone to mistakes and hindering extensibility. Thus, whenever a new service is developed interpreting a time series event in a different unit, a new event is introduced. However, this new event still encodes a vector of floating point values, effectively duplicating the event's content. This approach does not scale well with an expanding service ecosystem. To avoid this scenario, a slot carries the semantic description of the event's content, effectively separating content from interpretation. This enables services to share event definition schemata, while specifying varying interpretations.

To define a semantic-aware EDA, the following formalization is used:

Definition 2.3.4. Given an EDA $E = (S, \Omega, \Omega^*, \omega, \tau)$, E is called *semantic-aware* if an ontology $O = (C, H^C, R, H^R, I)$ exists such that

$$\forall \omega_i^* \in \Omega^* : \exists C_j \in C : (o_i \in \Omega : C_j)$$

The formalization states that for each instance ω_i of an operation, a concept in C exists that maps this operation to a the concept C_j in O.

In the next section, key concepts defined in the ontology for the semantic model will be presented.

Semantic Concepts

Key concepts for the PSE are modeled in the persistent ontology. Individuals of concepts are stored in the runtime ontology and are created by the *runtime system* to keep track of the current CMV system state. These individuals will not persist when the *runtime system* is shutdown. Exceptions are individuals that specify service invocations, e.g., subconcepts of *Workplace* or *Workflow*. These individuals describe the construction of specific CMV systems for workflows already defined by users in the *runtime system*. Figure 2.10 provides an overview of key concepts in the ontology. In the following, these key concepts are presented.

Service — The *Service* concept models all services of the PSE. Whenever a service is invoked, an individual of this concept is created in the ontology. Because a service can be invoked multiple times, its concept is associated with a *Workplace* and a *StartService* concept. Both concepts describe how a concrete service individual can be invoked. A service can be linked to multiple *Workplace* and *StartService* concepts to use different invocation parameters. For instance, a data provider that publishes a dataset can be invoked multiple times with invocation parameters pointing to different datasets. This way, services connecting to data providers are independent of the dataset's location. In Section 3.1.6 this property is used to compare two neural simulations using the same data provider service implementation with different datasets.

Workplace — The *Workplace* concept defines a place or platform a *Service* individual can be invoked. An object relation links a *Service* concept to the *Workplace* concept. In addition, the *Workplace* concept is linked to the *StartService* concept to model service invocation parameters.

StartService — The *StartService* concept models the starting procedure of a service. It provides two data properties, *hasStartCommand* and *hasCommandArguments* defining the invocation of a service. To run a service concurrently on the same *Workplace*, the service must not use the same endpoint. Thus, the data property *hasCommandArguments* points to a configuration file. Upon invoking a service, the *runtime system* passes the specified configuration file to the service which specifies the nett endpoint initialization. Individuals of the *StartService* concept are stored in the persistent ontology.

Workflow — The *Workflow* concept models a workflow. A workflow is described by a set of services and their connections and models all concrete workflows. To this end,

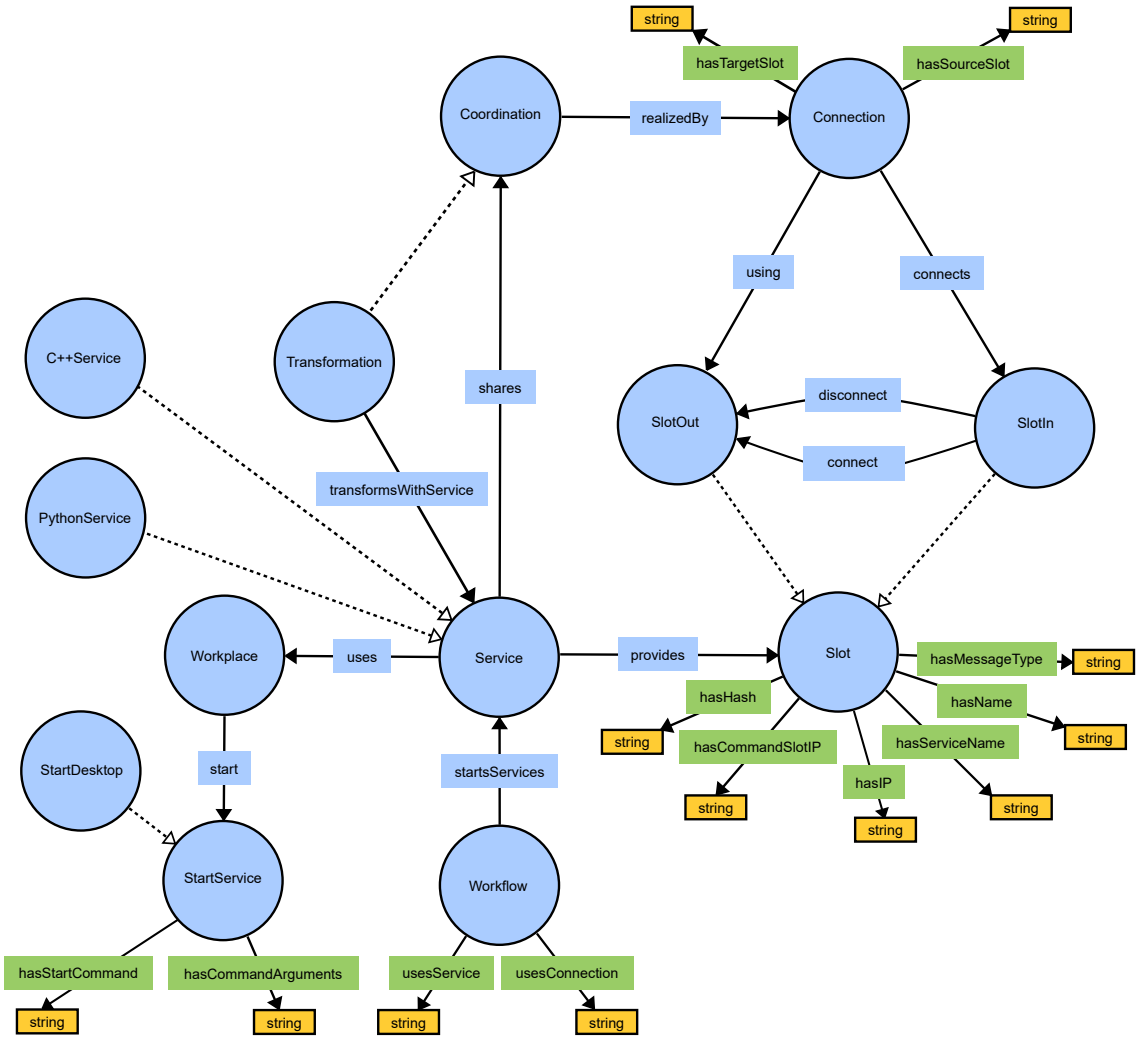


Figure 2.10: The semantic model used by the PSE consists of concepts and individuals. Concepts are marked as blue circles while arrows between concepts represent relations. A object property relation is indicated via a blue rectangular box. Data property relations are indicated as green boxes. Dashed arrows visualize a subconcept relationship.

the workflow concept enumerates on two distinct data properties: *usesService* and *usesConnection*. Thus, the concept models which services must be invoked and how they need be connected for an analysis task. To reproduce a workflow, the *runtime system* first iterates over all specified services of a workflow individual and starts these. Then, all connections are traversed and services notified to form these connections. Workflow individuals can be created by users once a CMV system is detailed within the PSE. Since the *runtime system* keeps track of services and their connections that form the current CMV system, it is possible to save this configuration as a workflow. Hence, for each analysis task a workflow individual can be created to avoid repetitive CMV system construction. Accessing and saving workflows is provided in the *runtime system* via its GUI. Individuals of the *Workflow* concept are stored in the persistent ontology to allow for reproducing the CMV system.

Connection — The *Connection* concept models a connection between slots. For each connection, an individual is created that identifies a connection by storing the source and target slot via its *hasTargetSlot* and *hasSourceSlot* data property. Since a slot is uniquely identifiable in the PSE via a hash value build over its semantic identifier, endpoint, and slot tag, both data properties relate to a string storing the hash value. A *Connection* is associated to the *Workflow* concept enabling its recreation. Connection individuals reside in the runtime ontology. Only if a connection individual is associated to a workflow, the connection individual is placed into the persistent ontology.

Slot — The *Slot* concept is the parent concept of all slot types. Because services communicate via slots, an object property *provides* relates a service to slots. As slots exist in two variants, this concept is subclassed by the concepts *SlotIn* and *SlotOut*. The slot concept models access to the service name it belongs to via a data property *hasServiceName* pointing to the service's individual in the runtime ontology. The slot's event type can be retrieved via the data property *hasMessageType*. The slot's semantic identifier is stored for each slot individual by the data property *hasName*. Its hash value, current network endpoint, and the service's control slot are accessible by additional data properties shown in Figure 2.10. Slot individuals are stored in the runtime ontology.

SlotOut — The *SlotOut* concept models publishing slots, also termed out-slots. Each individual of the *SlotOut* concept represents an available out-slot in the PSE.

SlotIn — The *SlotIn* concept represents invoked subscribing slots, also termed in-slots, in the PSE. For each in-slot of a service, an individual of the *SlotIn* concept is created in the runtime ontology. Because in-slots connect to out-slots to communicate events, this concept models two object properties, *connect* and *disconnect*. *SlotIn* individuals are allowed to create multiple connections to publishing slots resulting in the creation of corresponding *Connection* individuals.

Coordination — The *Coordination* concept specifies *Connection* individuals that are used to coordinate interaction among services that realize a CMV system. Since not all individuals of the *Connection* concept are used to coordinate interaction, e.g., connections to data providers, the *Coordination* concept allows to make this distinction. Individuals of the *Coordination* concept are identified via special semantic identifiers provided by CMV system services. Two examples of special semantic identifiers forming *Coordination* in-

dividuals are:

- *AreaSelection* which shares the current selection in views that support the selection of brain areas.
- *CurrentSimulationTime* coordinates the synchronization of simulation time in views that allow to navigate in time.

Chapter 3 will introduce further *Coordination* individuals based on the presented CMV systems.

Transformation — The *Transformation* concept allows to define services that can convert events to match different semantic identifiers. To this end, the *Transformation* concept defines a map of semantic identifiers and *proxy services* that can translate one semantic identifier to another one. Thus, transformations can be used to extend the PSE by *proxy services* that convert the output of out-slots of existing services to match input expectations of in-slots. Semantic inference then allows to find services to match a required semantic interpretation of an event. An example of this capability is presented in Section 2.3.5.

2.3.5 Event-Flow Network

An *event-flow network* is a graph consisting of all slots from invoked services and their connections. It defines the paths services exchange events. The construction of an event-flow network corresponds to the definition of a CMV system for a particular workflow and is assisted by the semantic model which imposes construction rules via its concepts and relations. The *runtime system* uses the semantic model to realize the construction of the event-flow network.

An event-flow network is formally defined as:

Definition 2.3.5. Given a semantic-aware EDA $E = (S, \Omega, \Omega^*, \omega, \tau)$ with an ontology $O = (C, H^C, R, H^R, I)$ as outlined in Definition 2.3.4, an event-flow network is a directed graph $G = (\Omega^*, M)$ where edges M are a set of ordered pairs defined as

$$M = \Omega^* \times \Omega^*$$

An event-flow network is dynamic because services can be invoked and add slots which inserts new vertices to the graph G. Moreover, connections can be established or revoked during runtime, thus changing the set M. A service itself does not correspond to a vertex in G, only its operations mapped to slots. Thus, each slot of a service represents a node in an event-flow network G.

Valid edges in G are defined by the following constraints:

Definition 2.3.6. For all ordered pairs $(\omega_i^*, \omega_j^*) \in M$, the pair is considered a valid edge if:

$$\omega_i^* \neq \omega_j^* \tag{i}$$

$$\omega_i^* : \text{SlotOut} \wedge \omega_j^* : \text{SlotIn} \tag{ii}$$

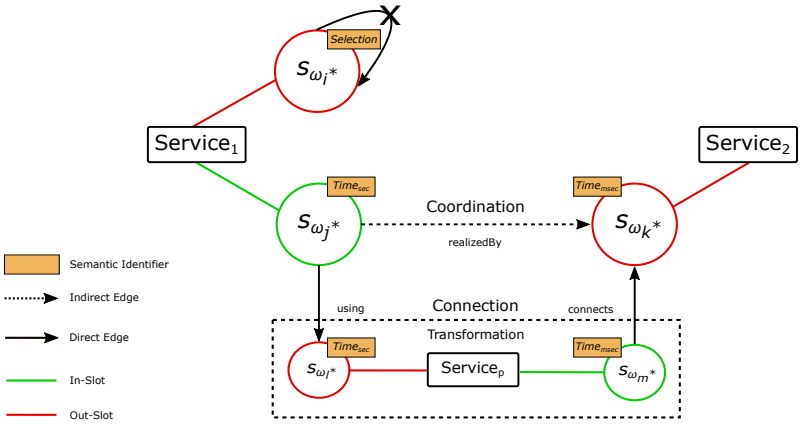


Figure 2.11: In this example, $service_1$ exposes an out-slot s_{ω_i*} and an in-slot s_{ω_j*}, which realizes a coordination with $service_2$ through s_{ω_k*}. A self-loop and direct edge of s_{ω_i*} to itself is not permitted, because an out-slot cannot connect to itself. In addition, a direct edge from s_{ω_j*} to s_{ω_k*} is invalid because the slot's semantic identifier do not match. However, the ontology defines a transformation so that a proxy slot s_{ω_l*} and s_{ω_m*} of $service_p$ can be used to transform the event's data and realize an indirect edge.

$$(\omega_i^*, c_i) : using \wedge (\omega_j^*, c_j) : connects \rightarrow c_i \equiv c_j, \text{ or} \tag{iii}$$

$$\exists a : \text{Transformation such that}$$
$$\exists c : \text{Connection}(a, c) : realizedBy \wedge (\omega_i^*, c_i) : using \wedge (w_j^*, c_j) : connects \tag{iv}$$

The first constraint states that no operation of a slot connects to itself. Thus, self-loops in G are not allowed. The second constraint ensures that only operations of in-slots connect to an operation mapped to a *SlotOut* concept. Constraint three requires that two operations participating in a coordination must be compatible by concepts defining the relations *using* and *connects*, i.e., in-slots and out-slots, as seen in Figure 2.10. Finally, the last constraint enables the transformation of an event's content into a different semantic interpretation through a *proxy service*. An edge constructed through this constraint is termed an *indirect* edge between two slots. To this end, there is a proxy service which transforms the output from an out-slot through a proxy in-slot, emitting the transformation to its proxy out-slot which is connected to the receiving in-slot. An example of an indirect edge is depicted in Figure 2.11 as a dashed arrow for a connection between s_{ω_j*} to s_{ω_k*}. All three constraints are enforced by the *runtime system*.

For example, a transformation can be useful given a service which emits time series data in milliseconds on its out-slot t_p and a service visualizing time series data in seconds from its in-slot t_c. A direct connection between t_p to t_c is possible, neglecting semantic constraints, because both slots exchange the same event type. However, the event's data interpretation is different, indicated by unrelated semantic identifiers. Nonetheless, using a transformation in the ontology, it is possible to find a proxy service for slot t_p that transforms the millisecond output to seconds for connecting t_c to t_p, thus introducing an indirect edge. If a connection of t_p to t_c is requested, the *runtime system* can infer that a connection can be realized by invoking a proxy service to realize an indirect edge between t_p to t_c. Transformations enable extensibility and flexibility for the PSE without the need to change existing implementations.

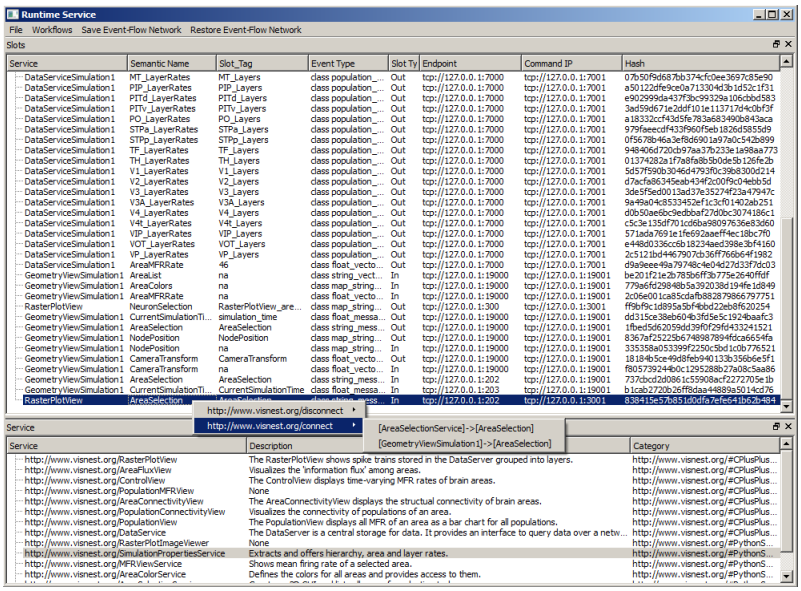


Figure 2.12: The *runtime system* shows all available services and slots via its GUI. By clicking on an in-slot, a connection to an out-slot can be established. Compatible out-slots are semantically inferred, ensuring that the connection is valid.

The construction of an event-flow network enforcing the semantic constraints is realized by the *runtime system* which will be discussed next.

2.3.6 Runtime System

The *runtime system* is the starting point to create a custom-tailored CMV system for an analysis workflow. It follows the idea that users select services depending on a workflow and connect slots, assisted by the semantic model, to reflect the workflow. To this end, the *runtime system* lists available services from the persistent ontology in a GUI for selection (see Figure 2.12 (bottom)). Subsequently, the *runtime system* realizes the event-flow network by connecting slots resembling the intended analysis workflow. Once an event-flow network is realized, the configuration and workflow can be saved.

The process to create a CMV system consists of the following steps:

1. start the *runtime system*.
2. invoke relevant services depending on a workflow or specify a new workflow by starting services.
3. construct an event-flow network by the means of semantic reasoning.
4. realize the event-flow network by connecting slots.
5. if the workflow of the user changes, repeat from step two.
6. when analysis concludes, shutdown services.

To allow for this, the *runtime system* provides two views: the first view, termed *slot view*, provides a list of announced slots from running services. The second view, called *service*

view, presents all available services from the persistent ontology. A drop down menu gives access to already established workflows to recreate a CMV system.

The *runtime system's* responsibilities are the invocation of services, the collecting of announced service slots, the creation and management of concept individuals, and the realization of the event-flow network by connecting slots. In addition, it enforces the event-flow network constraints by the means of semantic reasoning and enables the creation and storing of new workflows.

The *runtime system* tracks running services and their connections in the runtime ontology. To this end, the *runtime system* monitors the creation of slots and generates individuals of the *Slot* concept. When a slot is announced, the *runtime system* matches the semantic identifier associated with the slot to a concept. It then creates an individual in the runtime ontology by evaluating f and the operator S from Definition 2.3.3. Analogously, when a service is invoked a corresponding service individual is created. This enables querying of individuals that meet specific semantic criteria using SPARQL. To create individuals of the *Slot* concept, a service announces each of its slot via the *nett-semantic* library. This announcement includes the slot's service name, its semantic identifier, the event type it operates on, and whether it is an in-slot or out-slot. Whenever a service is started, the *runtime system* updates its GUI to show available slots as shown in Figure 2.12 (top) based on the individuals in the runtime ontology.

The construction of the event-flow network is also tracked by the *runtime system*. Whenever a connection between two slots is created, the runtime ontology is extended by an individual of the *Connection* concept. On this basis, the *runtime system* is able to infer which connections can be established and guides users in wiring services to form an event-flow network reflecting the analysis workflow.

To create or revoke a connection between slots, a user selects an in-slot from the *runtime system's slot view*. Here, reconfiguration can be performed by issuing a connect or disconnect command to compatible out-slots, hence the event-flow network can be *dynamically* changed at runtime. When an in-slot is selected, the *runtime system* retrieves all compatible out-slots that adhere to Definition 2.3.6 by performing a SPARQL query. Based on the query results, a context menu is populated listing all compatible out-slots, allowing to establish or revoke a desired connection. Once an out-slot is selected, it is checked whether a connection individual already exists in the runtime ontology. If so, the connection can be revoked and the connection individual is removed from the ontology. Otherwise, a new connection individual is created and a connect command emitted to the selected service's *control slot*. This functionality is provided by the *nett-semantic* library. Moreover, this functionality also allows to script an event-flow network without using the *runtime system*.

To store a new workflow, an individual of the *Workflow* concept is created and subsequently added to the persistent ontology. Then, running services and connections formed by slots are attached to the individual as data properties. The *runtime system* populates its workflow drop-down menu based on all individuals of the *Workflow* concept stored in the persistent ontology.

A CMV system constructed through the PSE is formalized as:

Definition 2.3.7. Given a semantic-aware EDA $E = (S, \Omega, \Omega^*, \omega, \tau)$, an ontology $O = (C, H^C, R, H^R, I)$, and an realized event-flow network $G = (\Omega^*, M)$, a CMV system

of the PSE is defined as

$$PSE_{cmv} = (E, O, G)$$

Thus, a stored workflow in the *runtime system* is an alias to a concrete PSE_{cmv}.

The *runtime system*, implemented in Python, utilizes the nett-python and nett-semantic module. The Qt library[14] is used for the GUI while for parsing, updating, merging, and semantic inference of ontologies rdflib[15] is employed.

In summary, the *runtime system* allows for service invocation, the construction of event-flow networks using semantic reasoning, and the management of analysis tasks as workflows. Examples of creating specific CMV systems within the *runtime systems* are outlined in Chapter 3.

2.3.7 Coordination Model

The coordination model used in the semantic-aware EDA uses the event channel to exchange coordination among services forming a *global distributed coordination space*. The global distributed coordination space is composed of events used to synchronize a service's view. Services receive and transmit events in the global distributed coordination space by *coordination endpoints* realized as slot connections. A coordination endpoint is a slot with an associated semantic identifier that maps to a subconcept of the *Coordination* concept from the semantic ontology as introduced in Section 2.3.4. A coordination of a conceptual entity is then defined as a connection between to coordination endpoints which are tracked as a *Connection* individual in the runtime ontology. Consequently, the semantic model allows to identify events and connections which belong to the global distributed coordination space.

Events exchanged in the global distributed coordination space are subdivided into five categories:

- **Data subspace:** the data subspace consists of events that are used to retrieve data from *data providers*. Events in this subspace facilitate a uniform and consistent way for services to access data for visualization purposes.
- **Selection subspace:** the selection subspace comprises events used to coordinate selection in views among services. Typical selection events are conceptual entities currently in focus.
- **Geometry subspace:** the geometry subspace encompasses geometric properties of conceptual entities shared among views. For example, global transformations applied to spatial objects or a used-defined color schema.
- **Parameter-subspace:** this subspace categorizes events that define visualization attributes that need to be synchronized among services to guarantee a consistent visualization state. Examples of events in this subspace are thresholds applied for conceptual entities, visibility, or data ranges.

[14] `https://www.qt.io/`
[15] `https://github.com/RDFLib/rdflib`

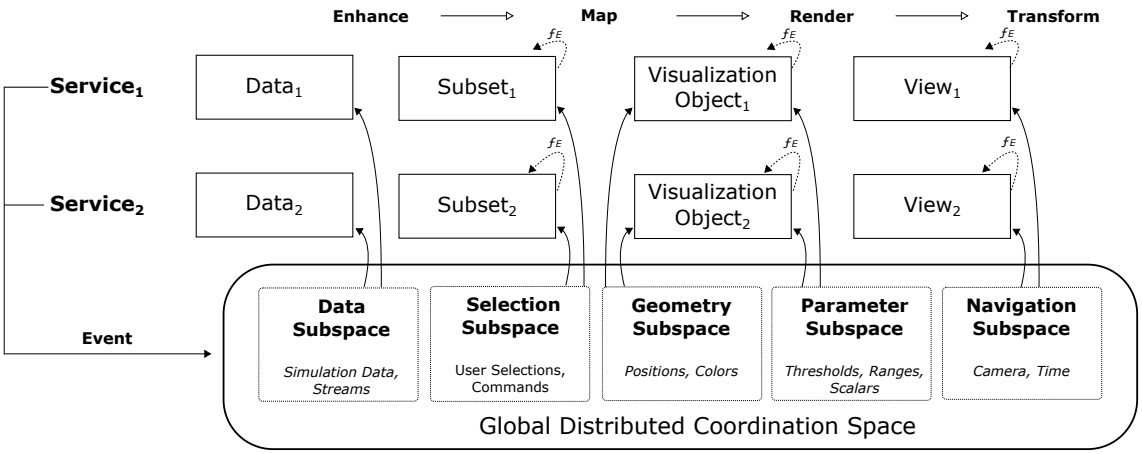


Figure 2.13: All coordination events form a global distributed coordination space. The coordination space is subdivided into five categories indicating the event's semantic interpretation. Services provide views depicting visualization objects as conceptual entities. User interaction publishes events to the coordination space. Once an event is received by a service, a transformation f is applied to react to the event's content.

- **Navigation subspace:** the navigation subspace covers events that coordinate visualization aspects like camera positing or current visualization time.

A unique property of this coordination model is its *dynamic* reconfigurability. Coordination can be requested and revoked at runtime through connecting or disconnecting coordination endpoints. This enables flexible scenarios where users can decide to coordinate conceptual entities based on current workflow needs. In addition, by adding new views to the PSE and extending the semantic vocabulary, new coordination mechanisms can be implemented without changing the coordination model.

The coordination model for the semantic-aware EDA is shown in Figure 2.13. Similar to the coordination model presented in Boukhelifa & Rodgers [2003] (see Figure 2.2), this model is designed without a bias toward data management or navigation concepts. It synchronizes views by utilizing the EDA's event channel and is also based on a data-flow model. However, while conceptually similar, several characteristics are special:

- a service embeds one or more views and provides basic communication facilities to access the global distributed coordination space.
- access to raw data to create, display, and manage a conceptual entity, i.e., visualization object, is defined as a coordination event between a data source and service.
- a coordination object, implemented via a coordination endpoint, facilitates the communication of coordination via a network connection.
- a service registers itself to the global distributed coordination space to receive notification of parameter changes.
- coordination endpoints are established over a network, enabling cross-device visualization systems and dynamic routing of events.
- coordination endpoints are modeled and managed in a semantic model. This allows semantic reasoning about the coordination endpoints' compatibility to establish co-

ordination.

- a service encapsulates a conceptual entity's *translation function* f to process an events received via the coordination endpoint. This allows services to define the interpretation of an event's coordination content.
- the semantic model, especially its capability to define event transformation as introduced in Section 2.3.4, allows for the extension of coordination concepts without modifying existing services.

In summary, the coordination model utilizes the event channel provided by the semantic-aware EDA. Coordination is realized by coordination endpoints connected to a global distributed coordination space where services can dynamically and remotely subscribe to. This enables CMV systems in HVEs. The global distributed coordination space is divided into five categories which identifies the coordination endpoint's purpose. Establishing connections between coordination endpoints is guided by a semantic model and is reflected in the event-flow network. The *runtime system* is used to establish coordination between views. Examples of coordination mechanisms, e.g., navigational slaving and linking, will be presented in Chapter 3.

2.3.8 Extending the PSE

This section presents, by way of example, how an additional service is added to the PSE.

Extending the PSE consists of four elementary steps:

1. the service implementation exposing of a set of slots.
2. extending the persistent ontology by adding an individual of the *Service* concept.
3. configuring the invocation of the service in the *runtime system*.
4. defining additional *Coordination* individuals used by slots.

To exemplify the extensibility of the PSE, a new service is developed to control the playback of visualization time. To this end, a GUI displays a slider to modify the current visualization time. To coordinate interaction with the slider, e.g., the user moves the slider back and forth, an out-slot publishes the slider's scalar value. To synchronize the visualization time modified from within other views an in-slot is added. Both slots use the same semantic identifier because they share the same semantic interpretation of the slider's value. In this example, the service is called *TimeController* while the semantic identifier for both slots is termed *VisualizationTime*.

Once the service implementation is complete, the persistent ontology is extended:

1. a new individual of the concept *Service* is created and named *TimeController*.
2. to start the service in the *runtime system*, an individual of the concept *StartService* is created.
3. the data property *hasStartCommand* for the new individual of *StartService* is set to point to the service executable.

4. the service individual *TimeContoller* is linked to the *StartService* individual to enable the *runtime system* to invoke the service.

5. because a slots is used to coordinate interaction, a *VisualizationTime* individual of the *Coordination* concept is added to the persistent ontology. This semantically differentiates a slot and indicates that its events participate in the global distributed coordination space.

After extending the persistent ontology, the *runtime system* will display the new service in the *service view* as *TimeController*. Once the *TimeController* is started, both of its slots are listed in the *slot view* as *VisualizationTime*. Here, the service's in-slot can be connected to any out-slot of a *Coordination* concept individual *VisualizationTime* for external updates of the time slider. Vice versa, the *TimeController's* out-slot is available to any view interested in coordinating the concept *VisualizationTime*. Finally, the service can be used in workflows requiring navigation in time.

CHAPTER 3

VISUALIZING NEURAL ACTIVITY DATA

The aim of computational neuroscience is to gain insight into the dynamics and functionality of the nervous system by modeling and simulating such systems. To this end, neuroscience leverages high performance computing to enable multi-scale simulations which capture low-level neural activity and large-scale interactions among brain regions. Higher-level brain functionality is based on the interaction of single neurons forming *neural networks*. How to construct, interlink and efficiently simulate these networks on a large-scale is a key research challenge in computational neuroscience [Plesser *et al.*, 2007]. The increase in computing power over the last decades originated efficient neural simulators like NEST [Gewaltig & Diesmann, 2007] or NEURON [Hines & Carnevale, 1997]. NEST is a simulator for networks of point neurons or neurons with few electrical compartments and runs on desktop computers, compute clusters, and large-scale supercomputers like the IBM Blue Gene/Q [Helias *et al.*, 2012]. However, these simulations produce a vast amount of output which needs to be analyzed to deduce the functionality of the simulated neural networks. In addition, neural simulations produce a multitude of data modalities, such as spike trains, connectivity data, and derived metrics on multiple scales, e.g., power spectrum, mean firing rate, etc. To this end, successful visualizations have to integrate this output into a unifying visual analysis tool, i.e., a PSE. While researchers validate findings by visualizing the simulation's output as part of a larger workflow, these visualizations are commonly non-interactive. However, state-of-the-art visualization techniques tailored to the scientific analysis question, but sufficiently general to accommodate different simulation models, enable such analyses to be performed more efficiently. Yet, the resulting heterogeneous data from simulations makes it hard to develop such visualization systems. Today's analysis workflows comprise a variety of different tools, ranging from standard shell commands to sophisticated analysis scripts, from simple descriptive statistics to elaborate correlation analyses. The amount of data combined with their complex inter-relationships demands a visualization system with a focus on *data integration*: various pieces of information at different scales and levels of abstraction have to be integrated into a holistic and configurable system that reflects the analysis workflow. A PSE enables data integration while CMV techniques assist in the analysis of the integrated data.

To demonstrate how the PSE from Chapter 2 is applied in this context, its applicability to cover the data integration task and to provide a highly customizable visualization system

is presented here. To this end, this chapter introduces three use cases assisted by interactive visualizations: three CMVs systems are derived which extend the PSE by services that provide visualizations.

The first use case will present views, conceived in close collaboration with domain experts, to analyze a neural network mimicking the visual cortex of a macaque. It focuses on the exploration of neural activity data, inspecting connectivity of brain areas and populations, and visualizing activity flux across regions [Nowke *et al.*, 2013, 2015]. The second use case will focus on data integration by extending the PSE with statistical analysis capabilities for spike trains. In addition to commonly used visualizations for the analysis of spike trains, this use case will demonstrate the flexibility of the PSE to integrate a community tool from the neuroscience domain. Finally, the third use case will present views and UIs to interactively steer a structural plasticity simulation. This allows for the interactive exploration and tuning of the neural network to detect parameter spaces that achieve a stable regime of neural activity [Nowke *et al.*, 2018]. This use case demonstrates how the PSE integrates data resulting from a running simulation and how users interact with it.

All use cases will extend the PSE by adding new services. Depending on the use case, one or more CMV systems are constructed, demonstrating a configurable visualization system that reuses services from other use cases.

3.1 Visualizing a Model of the Macaque Visual Cortex

3.1.1 Motivation

In this use case, several CMVs are presented for the simulation of a macaque visual cortex model. The CMV system helps in the assessment of the neural model presented in Schmidt *et al.* [2014] by visualizing its *network dynamics*. As part of the analysis process, the simulation output is filtered, statistically evaluated, and partially transformed into visual representations that encode network dynamics. Subsequently, these dynamics are visualized to verify, among other things, if the simulation shows a realistic low-activity state. To explore the simulation results, several views were conceived in close collaboration between neuroscientists and visualization experts. To relate the heterogeneous data, coordination such as linking and navigational slaving are required. To this end, the visualization system applies the CMV paradigm and is build on top of the semantic architecture. Services developed in this context will be reused, pronouncing the architecture's flexibility.

The development of this CMV system is motivated by four key elements:

- solutions to assist researchers in understanding the macaque visual cortex model and its neural dynamics by means of visualization are lacking.
- the simulation produces heterogeneous data that needs to be related in order to investigate the observed neural dynamics.
- the analysis process rapidly changes focus, requiring to switch to visualizations that are more helpful.
- convey findings to a broader audience by producing animations for publications and presentations of interesting network dynamics.

While both topmost elements were previously identified as requirements for the semantic-aware architecture, some visualizations techniques exists in related literature. For example, Schmitt & Eipert [2012] present neuroVIISAS, a platform for the integration of data modalities for the analysis and simulation of biologically realistic neural systems. It provides data analysis capabilities to explore neural dynamics and allows to model network descriptions which can be exported to the NEST simulator. However, neuroVIISAS does not focus on a tight integration of analysis workflows and visualizations required for the macaque visual cortex model nor does it enable users to define coordination among views and lacks support for cross-device display systems. NeuroVIISAS focuses on integrating pre-simulation data to identify biologically consistent network connectivity. A further example is presented by Arsiwalla *et al.* [2015] who introduce *BrainX*^{3}, a large-scale simulation system for brain activity with real-time interaction capabilities. It builds upon the *iqr* neural simulator and enables real-time visualization of network dynamics during simulation. Moreover, it allows steering a simulation by inducing activity to network nodes or disconnecting entire neuron populations. An interface to MATLAB for statistical analysis is also provided. However, the visualization system is specifically tailored to the authors' use case and is not easily applicable to the macaque visual cortex model. Sousa & Aguiar [2014] introduce the simulation environment *NeuralSyns* to build, simulate, and visualize large spiking neural networks. NeuralSyns provides a GUI called *NetBuilder* for network development using visual programming to build and parameterize complex networks without handcrafting models. NeuralSyns focuses on the development and analysis of neural models but its visualizations are tailored to spiking neurons. However, the macaque visual cortex model requires visualizations of derived modalities, e.g., the exchange of activity among brain regions. Furthermore, coordination is required to synchronize user selections and navigation in time while multiple views are displayed.

Focusing on visualization techniques, Hernando *et al.* [2012] explore real-time rendering of large amounts of neurons based on reconstructing biologically correct morphologies. For the macaque visual cortex model, a realistic representation of individual neurons is not of interest. Instead, relating structure to dynamics at multiple scales, ranging from single neurons to brain regions, and displaying averaged quantities is in focus. Especially this characteristic motivates a visualization design that utilizes concepts from CMV. In context of VR and cross-device display systems, von Kapri *et al.* [2011] present a 3D visualization of cortical layers which displays neural activity by rendering spiking neurons and plotting membrane potential. In contrast to the PSE, this single view system neglects additional simulation data modalities, offers no flexibility to extend the visualization to different analysis workflows, and does not provide coordination capabilities. However, its visualization design was a starting point in the development of this use case's CMV system. The PSE's flexibility easily allows for the integration of such a visualization as an additional view.

A system with similar requirements as the PSE is presented in Kasiński *et al.* [2009]. The authors introduce a CMV system for the analysis of dynamical processes in large spiking neural networks focusing on a 3D network representation. Aspect and scale changes are supported in conjunction with 3D navigation techniques. Furthermore, several visualizations are tied together to form a consistent highly integrated interactive system. This work addresses some of the needs for an interactive analysis tool assisting neuroscientists. However, the macaque visual cortex use case focuses on a network representation at the macroscopic scale, including connectivity and exchange of activity among brain regions.

Additionally, due to the large-scale connectivity data of neurons in the macaque visual cortex model, a 3D network representation is unfeasible and too detailed.

Eichelbaum *et al.* [2010] present a framework to deal with changing analysis focus and distinct visualization techniques to provide an intuitive brain visualization tool. *OpenWalnut* offers interactive visualization with an extensive set of features, e.g., tensor imaging, fiber tracts and line integral convolution while preserving flexibility and expandability. In contrast to our PSE which targets neural simulation data, OpenWalnut is tailored to neural imaging techniques; nor does it focus on CMV or multiple display systems.

To convey findings to a broader audience, the macaque visual cortex model requires a CMV system that is tailored to the simulation model's data. To this end, readily available toolkits such as ParaView [Henderson, 2004] or VisIt [Childs *et al.*, 2011] are not applicable without heavy customization and lack CMV support.

In summary, while some visualization system do exist in the literature, non are easily applicable to this use case. The overarching challenge consists in the integration of visualizations in a holistic system that allows to select views for an analysis workflow. This selection must then be configurable to specify how views are coordinated. The PSE solves this issue by utilizing standalone views with the capability to synchronize data. A set of connected views forms a CMV system.

The next section presents the neuroscientific workflow to analyze the macaque visual cortex model in order to derive services to visualize the simulation data.

3.1.2 Workflow Analysis

To derive services that reflect the visual analysis tasks, the researchers' modeling workflow is observed to deduce the resulting data modalities and relationships between the data. A key observation is a rapidly shifting analysis workflow due to changing hypotheses about the data. This requires flexibility from a visualization system. To derive requirements for integrating visual analysis into these workflows, understanding the modeling process is required. To this end, domain experts were asked to elaborate on their research objectives and workflows when developing the macaque visual cortex model which mimics biological realistic network dynamics [Nowke *et al.*, 2013; Schmidt *et al.*, 2014; Schuecker *et al.*, 2015]. However, the described objectives and workflows have been formulated in a broader manner and thus share a wider applicability. A spiking neural network simulation is based on a mathematical formulation which forms the basis for studying its behavior, structure, and size.

Five main objectives for the macaque visual cortex model have been identified [Nowke *et al.*, 2015]:

1. **Formulate a consistent model definition (O1):** derive a consistent model definition based on anatomical and electrophysiological data. This data is gathered from publications and databases to achieve simulations in accordance to biological findings, e.g., Stephan *et al.* [2001a], Binzegger *et al.* [2004], and Markov *et al.* [2011].

2. **Systematical parameter study (O2):** systematically study the impact of parameters on the model dynamics and modify the connectivity within reasonable bounds to reach a stable ground state close to biological findings [Schuecker *et al.*, 2015].

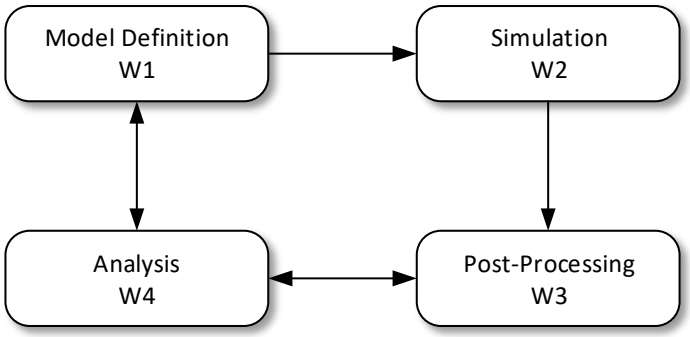


Figure 3.1: Depiction of the four elementary workflow steps: arrows indicate influence of a workflow step to another one. After the model definition (**W1**) is concluded, a simulation is performed (**W2**). Subsequently, the simulation output is post-processed (**W3**) and used in the analysis step (**W4**). Analysis influences the post-processing step whenever a new observable is derived. Analysis also effects the model definition, e.g., by changing the model's connectivity to converge to more biologically realistic activity patterns.

3. **Investigate emergent behavior (O3):** investigate the underlying mechanisms in differences in firing rates across populations and oscillations emerging through interactions between areas.

4. **Integrating scales (O4):** bridge the gap between large-scale models where each area is represented by a simple dynamical system and detailed spiking models of local cortical networks.

5. **Research scaling behavior (O5):** study the scaling effects by increasing the number of neurons up to realistic sizes of biological systems.

To incrementally improve the model, a workflow is established to systematically investigate the research objectives **O1**-**O5** (see Figure 3.1). This workflow consists of the definition of the simulation model (**W1**), its subsequent simulation (**W2**), a post-processing step of the resulting output (**W3**), and the assessment of the model's behavior by analyzing this output (**W4**):

1. **Model definition (W1):** the simulation model is implemented as a NEST script which can be developed in SLI, PyNEST [Eppler *et al.*, 2009; Zaytsev & Morrison, 2014], or PyNN [Davison *et al.*, 2009]. Configuration data is used to conduct parameter studies to produce changes in dynamics (**O2**). For instance, defining synaptic strengths, the relative strength of inhibitory compared to excitatory synapses, the connectivity of neurons, and stimuli to induce activity. To record data, virtual recording devices are assigned to neurons. Typical devices are spike detectors monitoring spiking activity, or voltmeters recording voltage traces. The devices' output is the raw data for investigating the behavior of the neural system. It forms the input for statistical analysis and visualizations (**O3**).

2. **Simulation (W2):** once the model definition has been completed, simulation is performed. Depending on its computational complexity, it is simulated on workstations or compute clusters. Reproducibility is ensured by storing the model defini-

tion and configuration data, e.g., in Sumatra [Davison, 2012]. This task is limited by computation time.

3. **Post-processing simulation output (W3):** when the simulation has been finished, aggregation of data from recording devices begins. NEST simulation output is scattered over multiple files. This requires merging simulation output to compute observables quantifying network behavior. These observables are used for visual parameter studies (**O2**) to converge to a consistent model definition (**O1**). the post-processed data are dependent on the simulation model and research question. For the multi-area macaque visual cortex model, connectivity data is included on multiple scales. The model consists of brain areas composed of populations of neurons. Computation of observables for areas and populations requires the model definition to investigate network dynamics (**O3**). In this use case, statistical observables are the areas' and populations' mean firing rate , spike trains, additional activity proxies, structural connectivity differences among populations, and correlations of spiking neurons.

4. **Analysis (W4):** data analysis is performed to assess the model's behavior (**O1, O4, O5**). The data analysis is augmented with visual data analysis techniques, in particular a CMV system to investigate **O2** and **O3**. However, in Section 3.3, a use case will be presented that targets **W2** and **W3** in conjunction with **W4**. As emphasized, this step is rapidly shifting focus and is not performed in a specific order. Hence, a flexible analysis framework based on the PSE is utilized.

Before utilizing interactive analysis, the typical workflow involved static figures to display observables. These figures were usually generated using several tools, e.g., Matlab or Python, resulting in laborious tasks for large networks. Notably, depictions were not linked, making it hard to relate information evident in one figure to aspects shown in another. Moreover, the analysis task was delayed by creating these figures and changing parameters required repeated executions. A CMV system overcomes these shortcomings and enables the reuse of visualizations in a wide range of workflows. Thus, a CMV system interactively and simultaneously displaying simulation output to reveal meaningful relationships is the primary motivation to improve the analysis workflow.

In the following section, the heterogeneous data resulting from the simulation model are discussed to derive visualization requirements.

3.1.3 Input Data Analysis

This section classifies the simulation data for the macaque visual cortex model resulting from the workflow steps **W2** and **W3**. This heterogeneous data was, for the most part, generated with NEST and forms the input data to the CMV system. The simulation model is subdivided into 32 areas according to Felleman & Van Essen [1991]. Each area contains eight neural populations, corresponding to excitatory and inhibitory neurons in each of four cortical layers (see Figure 3.2). Population-specific connectivity was carefully compiled based on anatomical records [Markov *et al.*, 2012; Stephan *et al.*, 2001b] and newly derived regularities. Intra-area connectivity is based on a model of the early sensory cortex by Potjans & Diesmann [2012] and has been adjusted to the macaque. A range of observables from population-level activity to inter-area interactions is computed resulting in nine heterogeneous data elements, also summarized in Table 3.1:

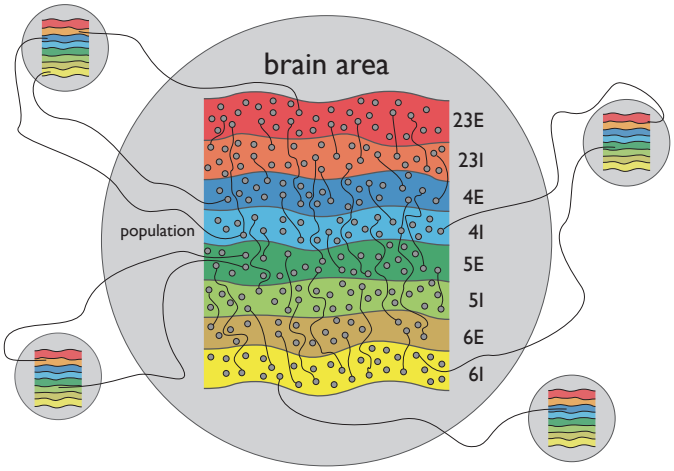


Figure 3.2: Schematic view of the input data: the dataset consists of 32 brain areas. Each area is a layered model of eight neuron populations. Connections exist at all levels, i.e., within the same population, across populations of the same area, and across areas.

1. **Spiking events (D1):** in experimental neuroscience, neural activity is obtained through electrodes implanted into brain tissue and is recorded while a stimulus is presented. In computational neuroscience, activity data is gathered by performing simulations and is then used as a verification baseline to approximate results close to its biological counterpart. The simulation yields spiking patterns of neurons over the simulation period.

2. **Mean firing rate (D2):** time-varying firing rates are computed at the population and area level by averaging spiking activity across neurons.

3. **Polygonal geometry (D3):** the geometry of the macaque visual areas was taken from the Scalable Brain Atlas [1] to visualize mean firing rate and provide a selection metaphor. It was reconstructed from volumetric data and relates activity data to geometry.

4. **Functional descriptions (D4):** a short description for each brain area and its role in visual function is provided. This places areas in a functional context.

5. **Area and population connectivity (D5)**: connectivity information is stored at both the area and population level. At each level, connectivity is represented in two ways: as mean numbers of synapses impinging onto individual neurons (in-degrees), and as mean numbers of synapses established by individual neurons (out-degrees).

6. **Hierarchy of the visual cortex (D6):** Felleman & Van Essen [1991] proposed a visual cortical hierarchy. This is used as a layout for arranging areas.

7. **Flux (D7):** the simulation yields synaptic activity per unit time between areas termed *activity flux*. The data contains 32×32 fluxes for each time step, including self-flux, i.e., activity exchanged within an area.

8. **Area and population names (D8):** a brain area consists of a set of population representing neural layers whereby a population is formed of a set of neurons. Areas

[1] `http://scalablebrainatlas.incf.org/`

Table 3.1: Classification for the macaque visual cortex model data.

Data	*Scale*	*Type*	*Origin*	*Label*
Spike trains	Microscopic	Time-varying scalar	W2	D1
Mean firing rate	Macroscopic	Time-varying scalar	W3	D2
Polygonal geometry	Macroscopic	3D geometry	W4	D3
Functional description	Macroscopic	Literal	W4	D4
Area names	Macroscopic	Literal list	W4	D8
Area colors	Macroscopic	Color list	W4	D9
Population names	Mesoscopic	Literal list	W4	D8
Population colors	Mesoscopic	Color list	W4	D9
Area connectivity	Macroscopic	Scalar	W1	D5
Population connectivity	Mesoscopic	Scalar	W1	D5
Hierarchy of areas	Macroscopic	Scalar	W1	D6
Area flux	Macroscopic	Time-varying scalar	W3	D7
Population flux	Mesoscopic	Time-varying scalar	W3	D7

and populations are labeled with literals representing their names. These literals are used to reference areas and populations more easily.

9. **Area and population colors (D9):** areas and populations are assigned consistent colors across views to more easily relate these.

In the next section, I will derive requirements for services that allow for the visual investigation of the presented research objectives based on the workflow needs and the heterogeneous data.

3.1.4 Requirements Analysis

In this section, functional requirements are presented based on Nowke *et al.* [2013, 2015] and were gathered in regular monthly meetings with domain experts. These meetings were used to assess development: identify, discuss, clarify, prioritize requirements, and suggest improvements to established functionality.

A central requirement is the support for visual data analysis in the exploratory data analysis process (**W4**). To this end, a CMV system is developed assisting in the analysis by displaying simulation data **W2**, including derived observables **W3**. Because of the high variety of analysis tasks, this CMV system has to be flexible, particularly in the analysis steps **W3** and **W4**. This aligns with the principles of the semantic-aware EDA to implement a custom-tailored CMV system populating the PSE. However, this results in granular requirements for services. For example, simulation data (**D1, D5, D6**) and derived observables (**D2, D7**) need to be accessible to views. Thus, access to data resulting from **W1**, **W2**, and **W3** is required in workflow step **W4**. Nonetheless, views can also provide data, i.e., coordination endpoints such as user selections or time navigation. Because of loose coupling between services, the configuration of the event-flow network should be supported within the modeling environment used in **W3**. This results in increased user acceptance and integrates into the model definition process (**W1**) and

post-processing (**W3**). The access to the *runtime system's* capabilities to manage and construct the event-flow network aims at a convenient way to setup views and modify data resulting from **W1-W3**. Because the PSE only allows data exchange via slots, interfacing views adheres to a simple protocol and loading data does not require devising file format specifications, thus limiting compatibility decay. This results in more flexibility, e.g., when new observables are introduced in **W1-W3**.

One challenge in utilizing CMV systems is the coordination of views. CMV systems often need coordinations that are hard to anticipate by visualization experts once the analysis tasks changes. Because view coordination depends on inter-related information, e.g., relations among conceptual entities displayed within one of the visual analysis task, the proposed CMV system supports *dynamic coordination*. Dynamic coordination allows to specify how a conceptual entity is linked to others at runtime in dependence on a workflow. Ideally, established coordinations are interchangeable while performing a visual analysis task and adapt to a variety of user needs. This thesis's key contribution enables user to dynamically define, control, and manage coordination of views.

An effective CMV system should also allow for use of a variety of display systems, depending on the visual analysis task and available input devices. For instance, when 3D spatial data is displayed, views may utilize CAVE-like systems that enhance spatial data representations [Laha *et al.*, 2012; Raja *et al.*, 2004] and make use of intuitive interaction metaphors not available in desktop setups.

The semantic-aware EDA conceptually covers all requirements, enables *dynamic coordination*, utilization of a variety of display systems (see Chapter 4), and supports multiple navigation techniques. Several views are based on the VR-enabled software stack ViSTA which is a platform-independent VR framework [Assenmacher & Kuhlen, 2008] allowing visualizations to seamlessly scale from desktop workstations to large CAVE-like VEs. VR combines stereoscopic presentation with direct 3D interaction, offering intuitive interaction with data. Previous collaborations in computational fluid dynamics have shown that immersive visualizations were beneficial for understanding complex 3D data [Hentschel *et al.*, 2008]. Also, Laha *et al.* [2012] support this anecdotal evidence with a formal evaluation of VR-based volume data exploration. Thus, utilizing the semantic-aware EDA combined with the ViSTA software stack can provide similar benefits in context of visualizing brain simulation data.

Next, more detailed functional requirements for views are outlined:

1. **Data storage (R1):** an appropriate storage concept captures simulation output aggregated from **W2**, model parameters, model-topology if available from step **W1**, and further derived statistically observables computed from **W3** to support **O1**, **O2**, and **O3**. Ideally, simulation results are deposited to enable, e.g., parameter comparison of the neural systems' behavior or investigation of the scaling behavior between downsized neuron models and biologically realistic ones (**O2**, **O4**, and **O5**). In addition, the storage concept is flexible and not tied to a particular simulation model. On top, it includes visualization specific data, e.g., geometry for rendering, color tables, or settings for view management.

2. **Data access (R2):** data aggregated in **R1** is accessible to services. Because views can be displayed on different display- and computer systems as part of **W4**, network access is required. Data access is restricted to slot communication.

3. **Data modification (R3):** modifications on data resulting from **W1**-**W4** can be performed by services, e.g., a running simulation, the modeling environment, or views, via slot communication. When data changes, slot communication provides notifies services to ensure consistent data distribution.

4. **Integrating simulation output data (R4):** simulation output, e.g., statistical observables derived from raw simulation data, are one of the key means to assess a model's behavior in **W4**. These observables constitute part of the heterogeneous data to visualize. However, their investigation is dependent on the analysis needs and are often computed on demand. To this end, a CMV system ideally supports the computation of statistical observables by automating **W3** and enabling flexible integration of such.

5. **Coordinating views (R5):** views are specialized to operate on and visualize a subset of heterogeneous data originating from all workflow steps **W1**-**W4**. To reveal relationships among conceptual entities formed from the heterogeneous data, multiple views are used. To explore relationships, coordination between views is required. Coordinating interactions includes, e.g., user selections to synchronize the display of statistical observables related to the selection, or navigation in space and time. While performing visual analysis as part of (**W4**), coordination information is generated in views and distributed in the *global distributed coordination space*.

6. **Managing views (R6):** managing views includes starting, configuring, and controlling slots of a view. Users can dynamically configure, control, and change views via the *runtime system*, effectively influencing the event-flow network of the CMV system. View setups can be saved as part of an workflow and restored later. Managing views can also be performed in the modeling environment by by exposing the *runtime system's* capabilities. This enables users to modify data in **W4**, create new data providers, and adapt views.

7. **Applying techniques from the neuroscientists' workflow (R7):** visual designs used in views shall utilize established neuroscientific analysis techniques to ease the transition to an interactive visualization system. This includes displaying raster plots, connectivity matrices, and plotting common observables such as mean firing rates.

8. **Visualizing simulation data (R8):** visualize the data modalities presented in Section 3.1.3. Conceptual entities formed from this data are to be related where possible. For instance, activity data of brain areas shall be related to a geometric representation to intuitively assess active and non-active areas. Connectivity among areas and populations should be visualized and simultaneously present activity exchange.

While core functionalities are already covered by the semantic-aware EDA, views and services need to be developed and integrated into the PSE. To this end, the following section details services to cover the functional requirements.

3.1.5 Views and Services

In this section, views for the analysis of neural activity data for the macaque visual cortex are presented and linked to requirements. To address the central challenge of data

integration, the visualization system leverages CMV to present information in its relevant context. The system consists of independent, reusable services which are also utilized in later use cases. These services are divided into two categories:

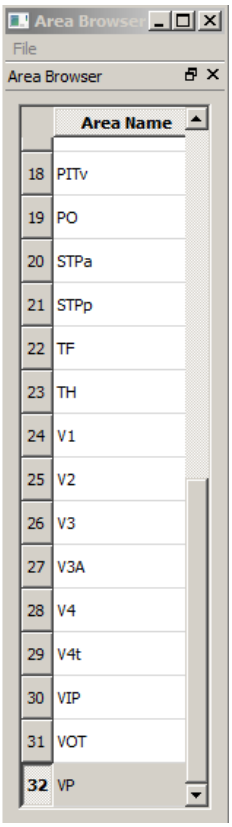
- **Data providers:** data providers are services that aggregate data from data sources and provide data access via slot communication by publishing it through out-slots. Data sources store content generated in workflow steps **W1**, **W2**, and **W3**. In this use case, data sources store the input data for the CMV system as described in Section 3.1.3. A data provider, once started, forms an individual of the *service concept* (cf. Section 2.3.4) in the runtime ontology. Analogously, a data provider's slots constructs individuals of the *slot concept*, describing each slot's semantic, thus enabling the *runtime system* to infer compatible slots to connect to. Data consumers retrieve data from providers by connecting in-slots to the provider's out-slots. Data providers facilitate requirements **R1** and **R2** and supply the input data for any data consumer. Multiple data consumers can retrieve data from the same data provider.
- **Data consumers:** data consumers are services, usually views, that transform data from data providers into visual representations of a conceptual entities and offer interaction with such, covering requirement **R8**. To access data (**R2**) data consumers subscribe to data providers. Similar to data providers, data consumers relate to individuals of the *service concept* and create individuals of their slots. This extends the runtime ontology (see Section 2.3). While data consumers can also produce and publish data to the *global distributed coordination space*, this light-weight data is used to realize requirements **R3**, **R5**, and **R6** but is not considered a *data provider*.

A service is unaware of any other connecting services and all service communication requires semantically described slots. As introduced in Section 2.3.6, the *runtime system* interfaces services by connecting semantic compatible slots, effectively describing a distributed event-flow network [Abram & Treinish, 1995], resulting in in a highly-decoupled and modular system. By connecting multiple subscribing slots to a publishing slot, multiple views are synchronized. To manage views as required by **R6**, each service exposes a *control slot* to receive commands it then executes. Examples for typical commands are to establish a connection to an coordination endpoint or to disconnect from it. Whenever slots are connected, a connection individual of the *connection concept* is created and stored in the *runtime ontology* to represent the current event-flow network.

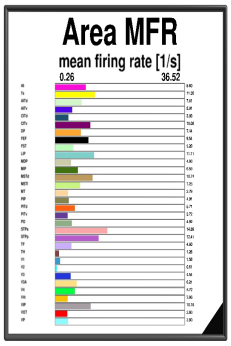
In the following, views for the macaque visual cortex CMV system are presented:

Simulation Data Provider — This service aggregates simulation data (**D1-D8**) from a file storing a simulation run of the macaque visual cortex model and is a *data provider*. The file is an HDF5 [Folk *et al.*, 1999] container assembled in workflow step **W3** and is loaded and published via slots (**R1**, **R2**, **R3**, **R4**, **R5**). The data contains the names of brain areas, populations, the simulation duration, spike trains for populations, the mean firing rates of areas and populations, flux, the connectivity and hierarchy used to simulate this brain model.

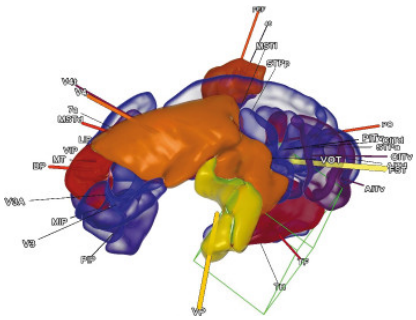
Color Service — This service exposes user-defined colors for areas and populations (**D9**) and acts as a *data provider* (**R1**, **R2**, **R6**). Colors are subsequently used to establish a unified color representation to relate areas and populations more easily. To retrieve colors, views connect to this service's slot.



(a) Area Selection View

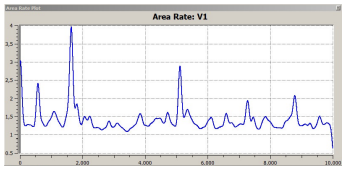


(b) Area MFR View

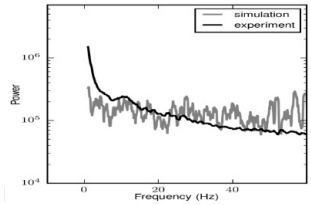


(c) Geometry View

(d) Time Navigator



(e) Time Series View



(f) Image Viewer

Figure 3.3: Collection of views: (a) The *Area Selection View* consists of a GUI displaying literal values as a list. User selections are published to synchronize selection events. (b) The *Area MFR View* shows areas and indicates area activity by moving bars. Next to each bar, the current activity level for the area is shown. (c) *Geometry View* showing a semi-transparent rendering of vision-related areas of the macaque brain. Coloring of brain areas is tied to their current activity. (d) VCR like interface to navigate through time. The slider's value is published and is used to coordinate time. (e) The *Time Series View* shows averaged bundled neuronal activity over a period of time for neurons of an area or population. (f) The *Image Viewer* allows to associate images to user selections.

Area Selection View — The *Area Selection View* provides a 2D GUI displaying brain areas as literals (**D8**) (see Figure 3.3a). Whenever a selection is made, the view publishes this selection. Multiple selections of brain areas are are allowed (**R5**). Moreover, the view receives selection events from other services and highlights corresponding entries. This view is used to select and filter brain areas and represents a *data provider*.

Time Navigator — The *Time Navigator* provides a 2D UI to navigate in time and is inspired by a VCR interface (see Figure 3.3d). The *Time Navigator* has a slider, a play and pause button, and a replay-speed controller slider. It publishes current simulation time and has a subscribing slot to synchronize time navigation events from different views (**R5**). It acts as a *data provider and consumer*. This service extends the *coordination concept* from Section 2.3.4 by creating a *CurrentSimulationTime* individual, thus semantically describing the coordination capabilities of this service.

Time Series View — The *Time Series View* provides the means to assess time series

data. It only consumes data and therefore is a *data consumer*. In this use case, it is used to plot the mean firing rate for a set of neurons (**D2**) (see Figure 3.3e). The *Time Series View* can superimpose a vertical line over the time series to indicate current simulation time. It uses standard techniques from the neuroscientists' workflow (**R7**) and integrates this simulation output (**R4**).

Image Viewer — The *Image Viewer* is a service and *data consumer* that displays generic images (see Figure 3.3f) related to user selections, e.g., from the *Area Selection View*. In this use case, the *Image Viewer* shows the power spectrum and rate distribution for the entire simulation run (**R4**).

Geometry View — The *Geometry View*, primarily a *data consumer* but also a *data provider*, forms the central starting point for the assessment of neural activity data (**R4**, **R8**). It provides access to user interactions that are subsequently used in other views, e.g., to allow for navigational slaving. The *Geometry View* displays 3D data representing the brain areas (**D3**) as seen in Figure 3.3c. This allows to visually explore spatial neural activity and their relationships. Activity data (**D2**) is projected onto each brain area similar to fMRI (**R7**, **R8**). The mapping of activity to area geometry is achieved by color-coding using a user-configurable lookup table. However, partial, mutual occlusion of brain areas and their complex shapes make it difficult to recognize individual areas depending on perspective. To this end, an area's activity level is also used to modify the area's opacity so that areas with lower activity do not overly occlude more active ones. To highlight the complex geometrical structure of the brain, an *angle-based transparency* technique is used similar to Hummel *et al.* [2010]. This illustrative approach modulates the α-value based on local view direction $\hat{v}$ and surface normal $\hat{n}$. This results in increased transparency of surface areas orthogonal to the direction of view, and decreased transparency where the surface of the area curves away from the view direction. However, the original approach provides no possibility to control opaqueness based on average spiking activity. To this end, an α-modulation technique which combines aspects of angle-based transparency and a mapping of activity data is used. let α_{base} denote the base transparency, and $\overline{a}$ the average spiking activity which is normalized to the unit interval over all simulation time steps. Then, the overall transparency α_{final} is calculated by:

$$a = \begin{cases} \alpha_{\text{base}} + \overline{a} & \text{if } (\alpha_{\text{base}} + \overline{a}) \leq 1, \\ 1 & \text{otherwise} \end{cases} \tag{3.1}$$

$$\alpha_2 = 1 - \|\hat{n} \cdot \hat{v}\| \tag{3.2}$$

$$\alpha_1 = a\mathrm{e}^{b\mathrm{e}^{ca}} \tag{3.3}$$

$$\alpha_{\text{final}} = \begin{cases} \alpha_1 + \alpha_2 & \text{if } \alpha_1 + \alpha_2 \leq 1, \\ 1 & \text{otherwise} \end{cases} \tag{3.4}$$

In Equation 3.1, two transparency values are added and clamped to 1. Equation 3.3 is given by the *Gompertz function* which incorporates the area's activity level a from Equation 3.1. Parameters b and c define an x-axis displacement and a growth rate but are not used to convey data values; they are empirically set to $b = -7$ and $c = -5$. The base transparency α_{base} in Equation 3.1 can be interactively adjusted to enable the inspection of currently invisible areas. Finally, both α-values are combined in Equation 3.4. For visual depth cues and lighting, the Phong illumination model is applied. To assure correct

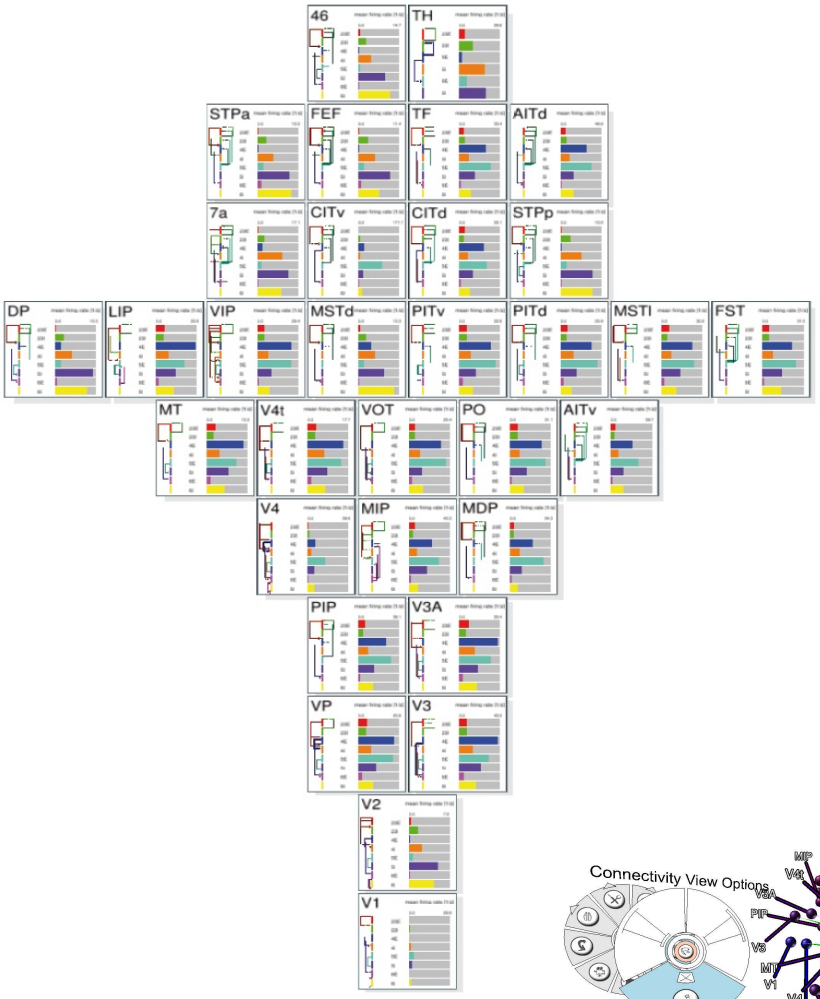
alpha blending, an order-independent transparency algorithm [Carpenter, 1984] is used. This order-independent transparency algorithm builds a linked list of (r,g,b,a,z)-tuples where z denotes the local depth value. After the rasterization pass, all fragments are sorted in depth-order and blended as shown in Figure 3.3c. In contrast to traditionally used activity plots, this depiction places the activity data in a geometric context assisting in the analysis of cross-area interactions.

To aid navigation, each brain area is annotated by its anatomical designator (**D8**). Because the view onto the scenery changes, annotation positions and their orientation are interactively adapted [Pick *et al.*, 2010]. The anatomical designators can be toggled to show the function description of the area in context of the visual cortex (**D4**). To reveal activity patterns in case of occluded areas, activity levels are also encoded via the connecting line between annotation and area. Its width is modulated with the local activity and its color chosen according to the activity color-coding. Moreover, the *Geometry View* allows to intuitively select and focus on brain areas to inspect these in more detail and coordinate this selection to subscribing views (**R5**). A selection is indicated by an outline of the area's bounding box. Users can freely position areas of interest with a dragging metaphor. The area's position are also exposed to coordinate areas' positions among other views (**R5**). To support temporal navigation in simulation time, e.g., starting or stopping the playback, the *Geometry View* provides an in-view VCR controller. This VCR controller is also exposed to allow for coordinating navigational time slaving (**R5**). Spatial navigation in the *Geometry View* is realized with a SpaceMouse or 6-DOF head-tracking. The user's position and orientation is exposed via slots (**R5**) to enable navigational slaving with different 3D views like the *Area Connectivity View* or the *Area Flux View*. Through navigational slaving as a linking technique, spatial context is preserved to relate visual entities that encode different data.

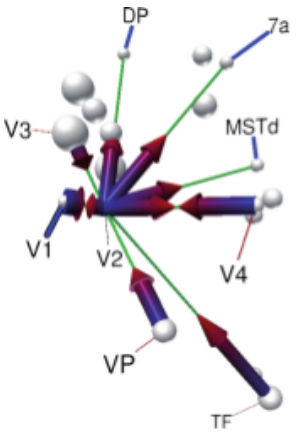
Area MFR View — The *Area MFR View* is a *data consumers* and *data provider*. It displays the mean firing rates (**R4**, **R8**), i.e., activity data (**D2**), of brain areas as moving bar charts as seen in Figure 3.3b and assists in the comparison of area activity. The moving bars use the color scheme from the *Color Service* (**R6**). In this view selections can be performed and coordinated with other views (**R5**) by hovering with a pointing device over a moving bar. In addition, selections performed in other views are highlighted. The *Area MFR View* requires coordination with the current simulation time to animate the moving bars. To this end, it connects to the *Time Navigator* or the *Geometry View's* in-view VCR controller (**R5**).

Area Connectivity View — The *Area Connectivity View* visualizes long-range connectivity (**D5**) among areas to assess structure and dynamics by displaying brain area connectivity (**R4**, **R8**). It acts as a *data provider* and *data consumer*. It uses 3D navigation techniques via a trackball metaphor utilizing a SpaceMouse, or 6-DOF head-tracking. Connectivity data is visualized as a weighted directed graph where each brain area is a node. A node is represented by a sphere and placed at the center of the area. This way, the geometrical relations with the *Geometry View* and *Area Flux View* are preserved. Node positions are coordinated among the *Geometry View*, and the *Area Flux View*. Whenever an area's position is moved, all coordinated views reflect the change (**R5**). A directed weighted edge is shown for a connection between any two areas. To avoid occlusion and clutter, the user selects an area for which all outgoing connections are displayed. The display of connectivity can be masked by a threshold range based on the *min*/*max* con-

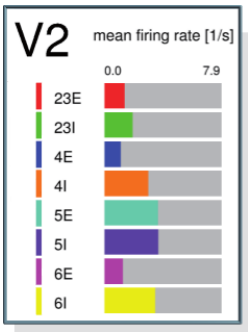
nection strength values and connections for specific areas can be turned off. Edges are drawn as straight lines with arrows depicting direction. Line thickness shows logarithmically scaled connection strength. Similar to the *Geometry View*, annotations depict the area's anatomical designator to aid navigation. The view implements navigational slaving (**R5**) by linking the user's position and orientation from the *Geometry View*. Figure 3.4d shows connections originating from area STPp. An extended pie menu [Gebhardt *et al.*, 2013] is used to control the view.



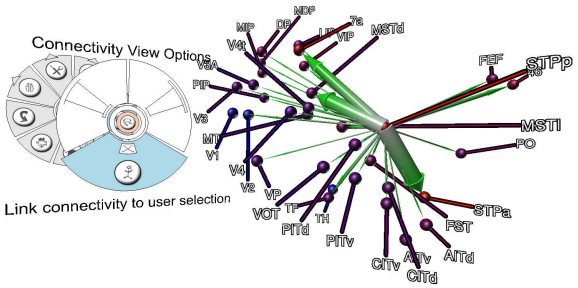
(a) Population Hierarchy View



(b) Area Flux View



(c) Population MFR View



(d) Connectivity View

Figure 3.4: (a) The *Population Hierarchy View* shows activity and connectivity where areas are arranged according to the visual cortical hierarchy proposed in Felleman & Van Essen [1991]. (b) The *Area Flux View* reveals a transient increase of the transmission from V1 to V2. (c) The *Population MFR View* visualizes the area's activity via an animated moving bar for all its populations. (d) The *Area Connection View* shows anatomical connections for a selected area. Thickness of edges encodes connection strength.

Area Flux View — The *Area Flux View* visualizes interactions between areas (**R8**). It acts as a *data provider* and *data consumer*. In combination with the *Area Connectivity View*, this visualization identifies large-scale communication patterns (**D7**). Interactions among brain areas can be studied in terms of numbers of spikes transmitted per unit time, a quantity called *activity flux*. Flux is represented as a graph where each node is a brain area. An edge consists of a time-varying scalar denoting changing edge weight. Translating activity flux into a visual representation is a four step process:

1. self-flux, i.e., activity within the same area, is obtained by extracting all edges with the same origin and destination.
2. inter-area flux, i.e., flux between two distinct areas, corresponds to all edges that are not self-flux.
3. both flux modalities are normalized to the unit interval.
4. flux is visualized as an animated graph (see Figure 3.4b) showing activity transfer.

Node positions are the areas' geometrical center position, represented as spheres, and coordinated among views to maintain context (**R5**). In addition, navigational slaving is supported by coordinating the user's view position and orientation from the *Geometry View* or *Area Connectivity View* (**R5**). In contrast to the *Area Connectivity View*, self-flux is mapped to an increase to the sphere's radius. To ensure visibility of areas with no self-flux, a minimum radius is defined. To avoid choppy animations in between time steps, linear interpolation is applied. Inter-area flux between two areas A_i and A_j is indicated by two color-coded arrows for each direction. To avoid overlap if flux is simultaneously exchanged, each arrow's length is limited to $0.5 \cdot d(A_i, A_j)$. Finally, flux is mapped to the arrow's thickness. To avoid visual clutter, flux can be masked by a *min* / *max* threshold or by filtering areas not in focus. Moreover, this view can also visualize flux between populations using predefined position for populations.

Population MFR View — The *Population MFR View* depicts each area as a panel and shows its populations activity data (**D2**, **R8**) similar to the *Area MFR View* to compare population activity. It requires coordination of selections from different views (**R5**) and acts as a *data consumer*. The panel displays an animated bar for each population's mean firing rate (see Figure 3.4c).

Population Hierarchy View — The *Population Hierarchy View* visualizes the mean firing rate of the areas' populations in conjunction with its hierarchy (**R4**, **R8**). It is a *data consumer* and *data provider*. The areas of the visual cortex form a hierarchy defined by laminar connection patterns identified in Felleman & Van Essen [1991] and are reflecting successive processing steps. Activation patterns can potentially be identified by visualizing areas according to such a hierarchy. To this end, the *Population Hierarchy View* provides an abstract hierarchical representation of all populations in each brain area (**D6**). Activity for individual populations is displayed the same as in the *Population MFR View*. The hierarchy layout resembles a *small-multiples* design (see Figure 3.4a). Each panel can be positioned freely for closer inspection. Selection of an area panel are also used to coordinate other views (**R5**). The coloring of bars uniquely identifies populations across multiple panels to detect regularities across areas (**R6**). Bars are stacked to the actual anatomical configuration of the layers (see Figure 3.2).

Population Connectivity View — The *Population Connectivity View* displays the connectivity of populations within each area (**R4**, **R8**). It is a specialization of the *Area Connectivity View* using the connectivity data of populations (**D5**) and is a *data provider* and *data consumer*. The *Population Connectivity View* facilitates the inspection and comparison of population-specific connection patterns influencing the dynamics of the network. Navigation supports a SpaceMouse or 6-DOF-head-tracking. To avoid visual clutter, dynamic range queries are supported where a *min* / *max* range is specified and connectivity

not within the range is culled. The *Population Connectivity View* also features a 2D visualization of population connectivity (**D2**). The visual design is influenced by the *Population MFR View* and connectivity data displayed as logarithmically scaled arrows between the population designators next to their moving bars (see Figure 3.5).

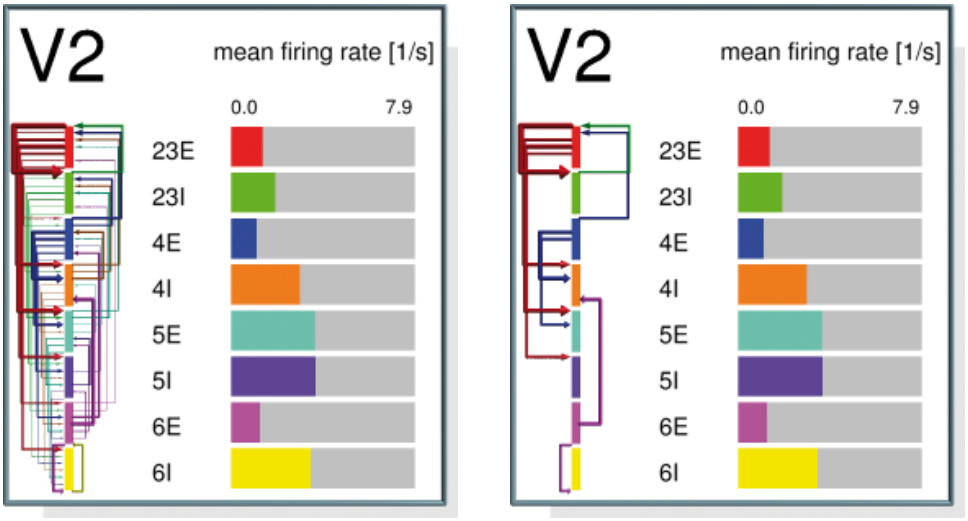


Figure 3.5: The *Population Connectivity View* shows connectivity of populations in an area. Left: Population connectivity without restricting connection strength. Right: The same connectivity with weak connections masked.

3.1.6 CMV System Construction

In this section, two workflows are presented to assess simulations of the macaque visual cortex model [Nowke *et al.*, 2013, 2015]. Each workflow requires a specialized CMV systems utilizing the previously presented views. The first workflow focuses on the assessment of *one* simulation run while the second workflow compares *three* simulation runs in parallel where model parameters, in particular the connectivity among populations and areas, were altered.

The CMV system construction includes the selection of services, their instantiation, and the creation of the event-flow network to reflect the workflow which is all performed in the *runtime system*. In semantic terms, both workflows are individuals of the *Workflow* concept (see Section 2.3) stored in the persistent ontology which allow to recreate the two CMV systems in the *runtime system*. In the following, creating the CMV systems for each use case is demonstrated.

Workflow 1: Single Simulation

For this workflow, a CMV system is constructed for the assessment of a single simulation run of the macaque visual cortex. Key research aspects for this task are:

1. **RQ1**: is a realistic low-activity network state present?
2. **RQ2**: are there interesting neural dynamics visible?
3. **RQ3**: is the model's activity stable?

4. **RQ4**: are there any unrealistic high firing rates in certain populations?
5. **RQ5**: does brain activity follow certain distributions?
6. **RQ6**: how does the simulation output relate to the neural dynamics of the model?
7. **RQ7**: are there any meaningful visible relationships?
8. **RQ8**: are there firing patterns present in populations?
9. **RQ9**: are there structural and functional relationships?
10. **RQ10**: are there pathways visible through time dependent activation of areas?

To assess the simulation output and extract insight, the CMV system is constructed from services. These services meet two functional criteria: *basic functionalities* and *research centric functionalities*. Services providing basic functionalities are the *Simulation Data Provider* to access the simulation data, the *Color Service* to retrieve a commonly-shared color schema for views, the *Area Selection View* to select and filter brain areas, and the *Time Navigator* to navigate in simulation time. The research centric functionalities are provided by services where each is linked to one or more research questions:

1. the *Geometry View* relates neural activity to compare areas. This view focuses on **RQ1, RQ2, RQ3, RQ5, RQ6**, and **RQ10**.
2. the *Area MFR View* showing accumulated area activity focusing on **RQ2, RQ3, RQ4, RQ5, RQ6, RQ8**, and **RQ10**.
3. the *Time Series View* visualizes the mean firing rate of an area, assisting in **RQ1, RQ2, RQ4, RQ6**, and **RQ8**.
4. the *Raster Plot View*[2] shows the spiking behavior of neurons for an area to focus on **RQ1, RQ2, RQ4**, and **RQ8**.
5. the *Area Connectivity View* visualizes connection strength among areas targeting **RQ7** and **RQ9**.
6. the *Area Flux View* displays activity exchange of areas, aiding in the assessment of **RQ1, RQ2, RQ3, RQ4, RQ5, RQ6, RQ9**, and **RQ10**.
7. the *Population MFR View* displays population-specific activity of an area to assess **RQ1, RQ2, RQ4, RQ5**, and **RQ8**.
8. the *Population Hierarchy View* identifies cascading activation patterns in focus for **RQ2, RQ4, RQ5, RQ9**, and **RQ10**.
9. the *Population Connectivity View* shows the connectivity of populations in an area, supporting **RQ3, RQ4, RQ6, RQ8**, and **RQ9**.

In particular research questions **RQ2, RQ5, RQ6, RQ7, RQ9, RQ10** are ideal candidates to utilize CMVs to reveal relationships.

The next phase of constructing the CMV system consists of starting all required services and establishing the event-flow network. To this end, the *runtime system* is used to start all services which in turn expose all their *coordination endpoints* by announcing their semantic slots. The *runtime system* then allows to connect semantic compatible *coordination*

[2] outlined in Section 3.2.5

endpoints to realize the event-flow network. Once an event-flow network is established, an individual of the *Workflow* concept stores this network in the persistent ontology to recreate the CMV system concluding its construction.

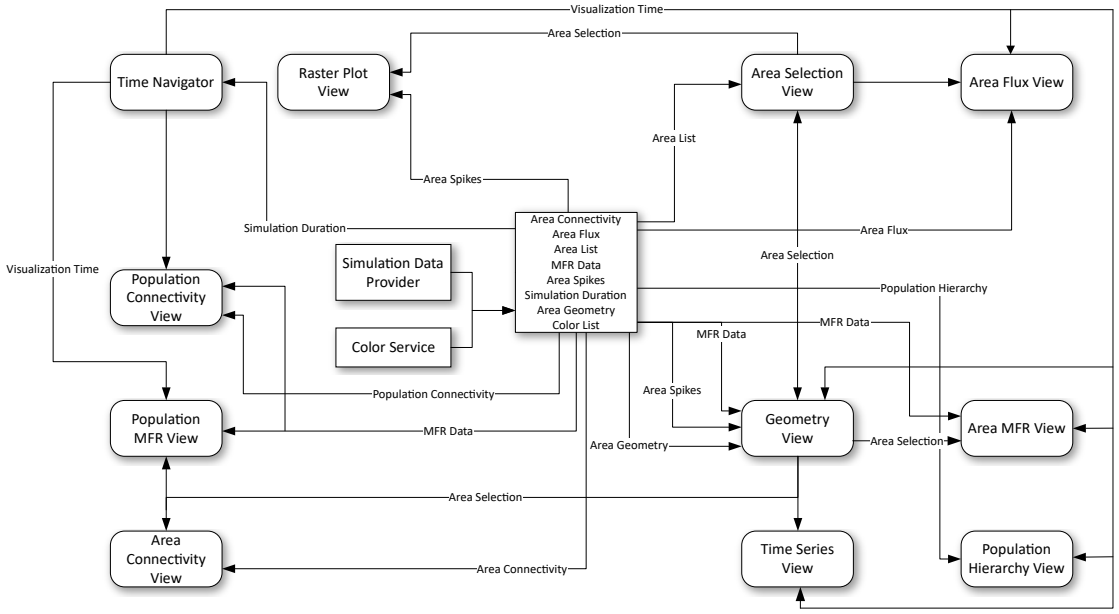


Figure 3.6: The event-flow network resulting from the coordination of views and services for evaluating a single simulation: slot connections are formed as indicated by arrows between services to communicate data. Communication via slots is unidirectional. Hence, multiple connections to slots of a service are formed. Multiple views are synchronized, e.g., the Area Selection to forward user interaction in views currently utilized by the user.

The event-flow network of the CMV system for this use case is depicted in Figure 3.6. Here, the *Simulation Data Provider* and *Color Service* provide the basic input data for the visualization services. These connect to the *Simulation Data Provider* and *Color Service*. Once service connections have been established, coordination is realized via receiving events. In this use case, the *Area Selection View* provides one mechanism to focus on a brain area. However, the *Geometry View* also allows to perform area selection. To this end, multiple services, e.g., the *Raster Plot View*, the *Area Flux View*, or the *Time Series View* which all operate on area selections, connect to views providing area selection capabilities. Analogously, visualization time is coordinated among services. The *Time Navigator* retrieves the simulation duration from the *Simulation Data Provider* and then computes visualization time to periodically publish it to its *visualization time* coordination endpoint. Via a VCR interaction metaphor, visualization time can be controlled to slow the playback or to seek to a specific point in the simulation. The *Geometry View* and *Area Flux View*, among others, are connected to the *visualization time* coordination slot and are synchronized with the *Time Navigator*.

This modular approach of EDAs allows to configure and scale the desired CMV system to different use cases. Moreover, views can be deployed on a diversity of display systems as shown in Figure 3.7.

While this CMV system is created to primarily assess a single simulation, the research objectives when comparing multiple simulation mostly stay the same where some research

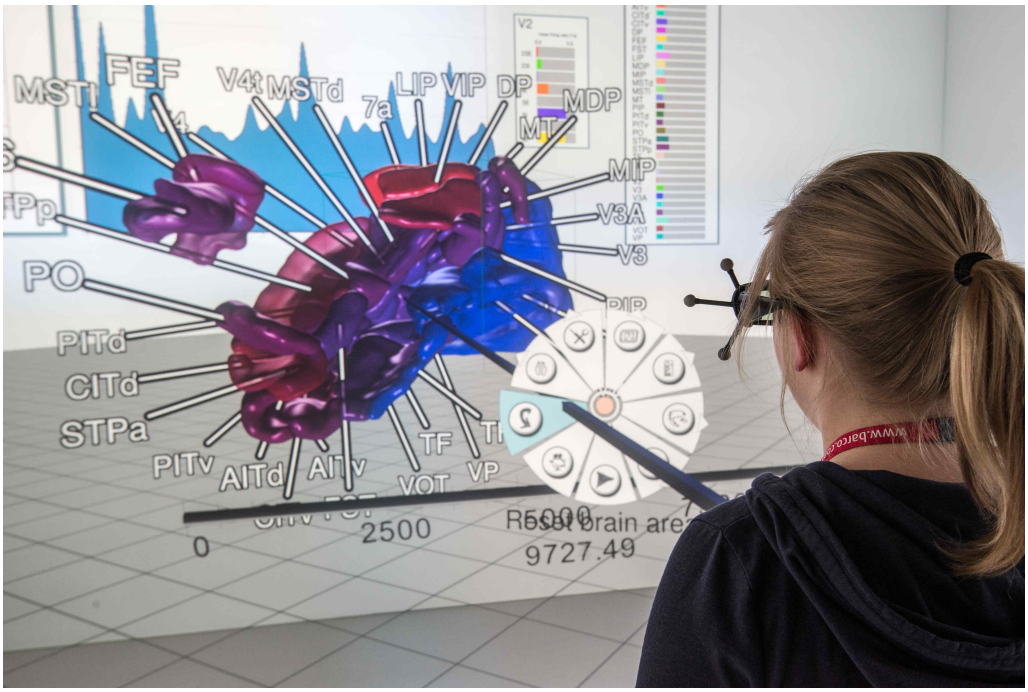


Figure 3.7: A user in a CAVE inspects a simulation using the *Geometry View*. Annotations tied to brain areas relate areas to geometry. Areas are color-coded with regards to their current mean firing rate. 2D panels (background) show statistical measures. System control is realized using extended pie menus [Gebhardt *et al.*, 2013].

questions shift focus. To this end, the next section introduces the workflow and its resulting event-flow network for comparing three simulations with different parameter settings, before an overall result discussion is presented.

Workflow 2: Comparing Simulations

In this use case, a CMV system is constructed to simultaneously compare three simulations of the macaque visual cortex model with different parameter settings. Key research aspects for this task are:

- **RQ11**: what is the basic influence of parameters between simulations?
- **RQ12**: what is the parameter impact on the network dynamics?
- **RQ13**: are specific activity distributions visible across areas?
- **RQ14**: can regimes in the cortical network be identified that show asynchronous firing behavior?
- **RQ15**: are contrasting interactions on different time scales present in one or more simulations?
- **RQ16**: is activity propagation present and are these similarly pronounced between simulations?

To visually compare the simulation output, services from the PSE are selected. In comparison to the first use case, the selection of services focuses on activity levels in more

detail. To this end, same services are invoked multiple times to concurrently visualize three simulation datasets. The resulting CMV system uses the following services:

1. the *Geometry View* is started three times to visualize the neural activity to assess **RQ11, RQ12, RQ13, RQ15** and **RQ16**.
2. the *Area Selection View* to select and filter brain areas across all views.
3. the *Time Series View* is invoked for each simulation to show mean firing rates of selected areas.
4. the *Image Viewer Service* is used six times to depict each simulation's firing distribution and power spectrum (**RQ13, RQ15**).
5. the *Population MFR View* is started three times to assess population activity (**RQ1, RQ2, RQ4, RQ5, RQ8, RQ15** and **RQ16**).
6. the *Raster Plot View* is invoked for each simulation to assess **RQ1, RQ2, RQ4, RQ8**, and **RQ14**.
7. the *Time Navigator* is used to navigate visualization time for all three simulations**RQ15**.
8. the *Simulation Data Provider* is started three times to access the simulation datasets.
9. the *Color Service* is used to retrieve the shared color-schema among all visualizations.

To compare the activity data of all simulations, a normalization of activity is required: activity is scaled using the natural logarithm over the *min* / *max* mean firing rate values of all simulations. This post-processing step is part of the workflow phase **W3** as described in Section 3.1.2.

The semantic-aware EDA provides the flexibility to concurrently use services with distinct configurations which makes it possible to link and coordinate views operating on different datasets. Thus, to compare simulations, the idea is to build one event-flow network consisting of three sub-event-flow networks which focus on the exploration of one simulation. These sub-event-flow networks then coordinate user interaction among each other to focus on the same visual entity using the different datasets from of each simulation. Lastly, the entire event-flow network is constructed in *runtime system* and saved in the ontology as an individual of the *Workflow* concept. The resulting event-flow network containing the three sub-event-flow networks is depicted in Figure 3.8 and the CMV system is shown in Figure 3.9. Similar to the first use case, each *Simulation Data Provider* provides access to the simulation data. However, only one *Color Service* is required for the a shared color-schema across all views. Analogously, only one *Area Selection View* (see Figure 3.9, top left) is used to select areas across all views. However, all *Geometry Views* synchronize selections: *area selection* endpoints are connected to all *Geometry Views* because an area selection can be performed at any time in one of the *Geometry Views* and must be coordinated among all views operating on an area selection. This pattern is applied multiple times: the *Time Navigator* connects to all *visualization time* endpoints from each *Geometry View* to realize navigational slaving. Also, the *Raster Plot Views* and *Population MFR Views* connect to the *Area Selection View* and to all *Geometry Views' area selection* endpoints. In similar fashion, area geometry positions and orienta-

tions are coordinated among the *Geometry Views* to synchronize when an area is dragged closer for inspection.

This use case exemplifies the modular approach of EDAs to build, scale, and tailor a CMV system. The complexity of developing a single CMV system is moved to the construction of event-flow networks but enables the reuse of existing views, making it possible to cover a vast amount of analysis tasks. In the following section, the CMV systems for both use cases are used to asses the simulation output with respect to the posed research questions.

3.1.7 Results

This section demonstrates the CMV systems and presents results obtained by domain scientists performing visual analysis.

After choosing the workflow in the *runtime system* from the stored individuals of the *Workflow* concept and constructing the event-flow network, investigation of the simulation can begin. In the first use case, an in-depth assessment of the neural behavior is performed when a realistic low-activity network state is observed (**RQ1**) where an analysis of simulation parameters on large-scale activity is in focus since not all parameter configurations yield realistic network behavior **RQ2, RQ3, RQ4**. This is partly due to the fact that available biological data are not sufficiently restrictive to completely specify the model and to some extent its inherent simplifications. Hence, to obtain realistic network dynamics (**RQ1**) and to gain an understanding of the underlying mechanisms, it is important to study the influence of parameter settings on the network dynamics. To this end, the *Geometry View* is used for a rapid overview of the large-scale network dynamics, including its stability (**RQ3**) and distribution of firing rates (**RQ5**). Especially for the second use case, this rapid assessment facilitates an intuitive visual comparison of the simulations. An example is presented in Figure 3.10 where external inputs and relative inhibitory synaptic strengths were changed between simulation runs (**RQ6**) and found to only yield realistic activity for one of the settings (**RQ1**).

Brain activity has a specific distribution (**RQ5**), not only across areas but also across layers and populations [Potjans & Diesmann, 2012; Shinomoto *et al.*, 2009]. For instance, inhibitory populations usually have higher firing rates than excitatory ones. This can be assessed with the *Population MFR View* which shows the impact of parameter settings on population-specific activity (see Figure 3.11). Other types of unrealistic activity, such as pathologically high firing rates (**RQ4**) or strongly oscillatory behavior (**RQ8**), can also be quickly identified in this view. For both examples, the previous workflow forced domain experts to manually generate these images; even the straightforward mapping of activity data to area geometries would not have been a standard operation. Moreover, the resulting stills would not have provided interactive access to the temporal evolution of the underlying data.

Another important investigation is the assessment of detailed firing patterns of populations within an area (**RQ8**). When modeling cortical networks, one usually aims to achieve a regime where single neurons emit spikes irregularly and groups of neurons fire asynchronously. To gain an overview of these detailed firing patterns, one can select these in the *Geometry View* or the *Area Selection View* to coordinate the corresponding *Raster*

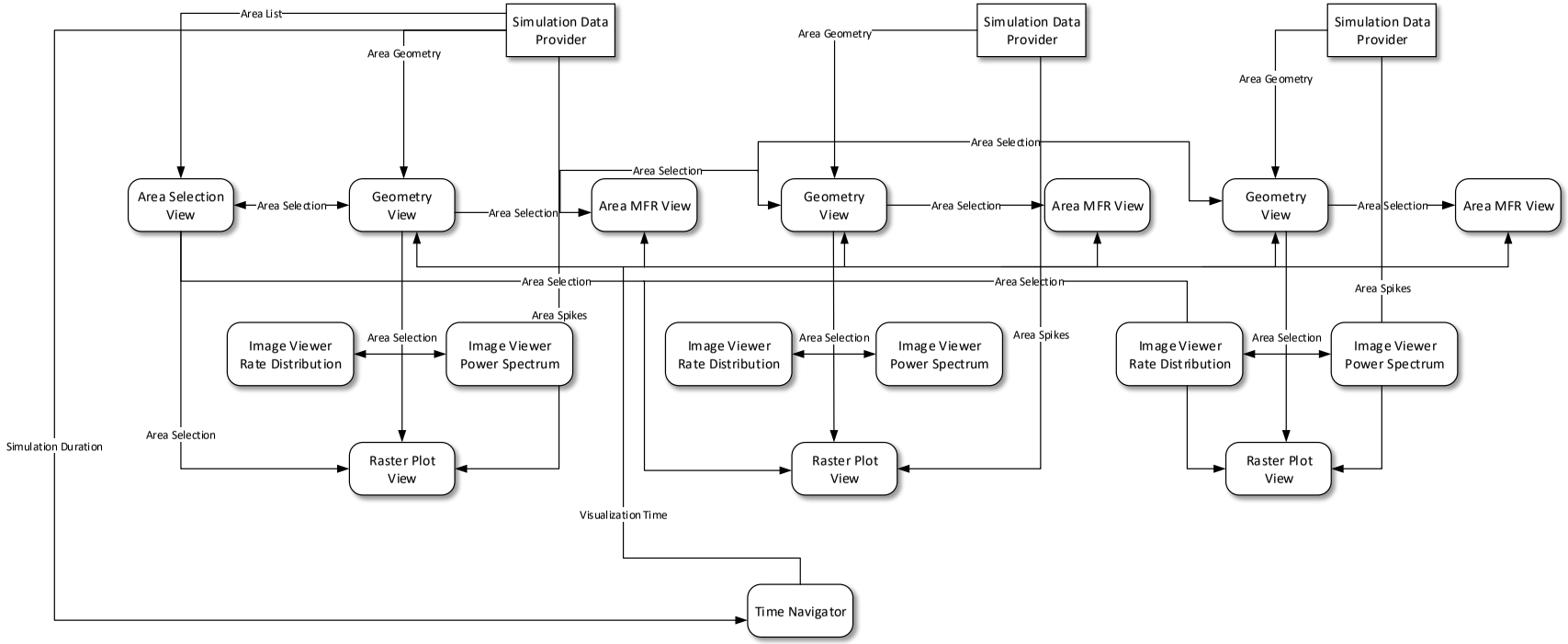


Figure 3.8: The event-flow network for comparing three simulations: slot connections are formed as indicated by arrows. Multiple area selection endpoints are connected between the *Geometry View* to synchronize user interaction in the CMV system. The visualization time is coordinated among services as well as area selections from the *Area Selection View*.

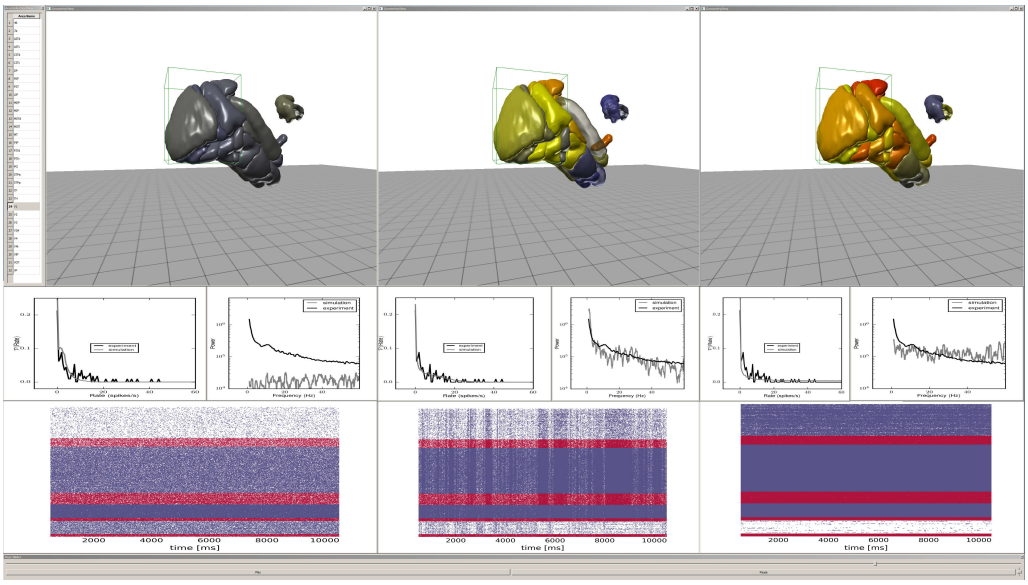


Figure 3.9: Comparing three simulations of the macaque visual cortex model yielding different activity levels due to connection-strength differences among brain areas. The *Area Selection View* is arranged to the top left and shows the selection of area V1. Next, three *Geometry Views* show the mean firing rates for each simulation and the current selection is indicated by a bounding box around area V1. At the bottom, for each simulation, two *Image Viewers* depict the firing distribution and power spectrum, followed by the *Raster Plot View* showing V1's dot plot. At the bottom corner, a single *Time Navigator* coordinates the visualization time for all views.

Plot Views. Here, an advantage over with static depictions is that the animated geometric representation enables the quick identification of areas based on their overall activity or co-activation with other ones (**RQ7, RQ8, RQ10**). Details can be accessed immediately, e.g., by focusing on the raster plot for an arbitrary area, no longer requires manual intervention and provides immediate feedback.

A further analysis task is the comparison of structural and functional relationships (**RQ9**). One aim of the visual cortex study is to understand the relationship between the cortical structure and its dynamics. The comparison of structural connectivity and functional relationships is possible through the *Area Connectivity View*, and the *Population Connectivity View*, and the *Area Flux View*. Comparing Figure 3.12 (left) and Figure 3.12 (center and right), one can see how the activity follows the structural connectivity. Here, only the flux via the strongest connection from area V1 to V2 is not masked by an threshold, filtering low synaptic exchange. This interactive comparison of structural and dynamical relationships between areas was previously not possible (**RQ9**). The functional interactions also depend on population-specific connection patterns which are visualized in the *Population Connectivity View*. It enables a direct comparison between detailed connectivity and the population-level activity. For instance, Figure 3.5 shows fairly strong connectivity from layer 4 to layer 2/3 as also seen in the connectivity matrix. The animated bar chart reveals that layer 4 activation tends to be followed by an increase in layer 2/3 activity, an observation that was complicated to make without interactive visual analysis (**RQ2**). Moreover, the control of visualization speed through the *Time Navigator* enables neuroscientists to observe interactions on different time scales.

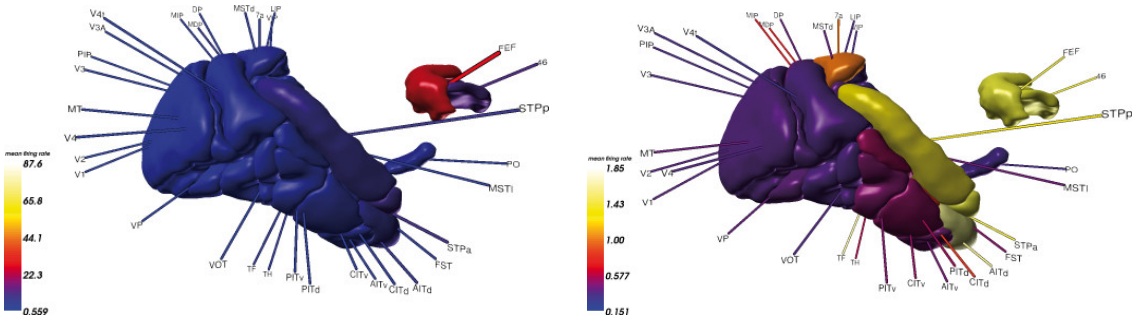


Figure 3.10: Comparison of two simulations using the *Geometry View*. In the simulation on the left, a few areas are highly active while the rest is nearly silent. On the right, a more realistic distribution of activity across areas is observed.

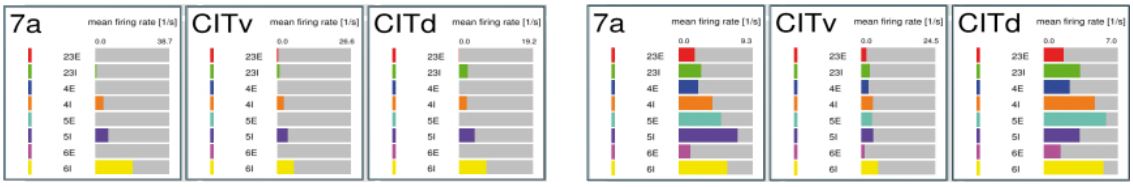


Figure 3.11: Comparison of two simulations using the *Population MFR View*. On the left, the excitatory populations are silent despite reasonable activity on the area level. On the right, both excitatory and inhibitory populations display activity conforming more closely to the biology.

The analysis of pathways (**RQ10**) is important because the perception of visual stimuli relies on time-dependent activation of areas [Lamme *et al.*, 1998]. Studying the spread of activity upon transient external stimulation is therefore important for understanding visual cortical function. Area *V1* is the main input station of the visual cortex activated by the thalamus upon visual stimulation. Figure 3.12 (center and right) shows how the transient activation of area *V1* yields a temporary increase in its outgoing flux, particularly to *V2*. Observing this activation in an interactive view aids studying these processes, particularly in combination with the control of visualization speed. This propagation of activity can also be followed in the *Population Hierarchy View* as seen in Figure 3.4a. The resolution at the level of single populations allows activities to be related to modeled population-specific connection patterns. For instance, projections from early stages of visual processing, e.g., V1 to higher areas, preferentially originate in the upper layers and terminate in layer 4 [Felleman & Van Essen, 1991]. Thus, the match between known anatomy and patterns like upper layer activation in a lower-order area followed by layer 4 activation at the next hierarchical level can be investigated in this view.

The second use case does not assess the simulations in as much detailed as the first. The CMV system for the comparison task focuses on the assessment of changing model parameters (**RQ11**), in particular the connectivity settings (**RQ12**). Because the visualization time and perspective on the scene in all *Geometry Views* are coordinated, the activity levels of areas can easily be compared (**RQ12, RQ13**). Also, when focusing on an area, all raster plots for the three simulated areas are displayed in the *Raster Plot View*, allowing to assess difference in firing patterns (**RQ14**) (see Figure 3.13). The *Population MFR Views* and *Time Series Views* allow to compare the activity levels in all simulations at once. The detailed view on population activity levels reveals activation patterns (**RQ13**) similar to

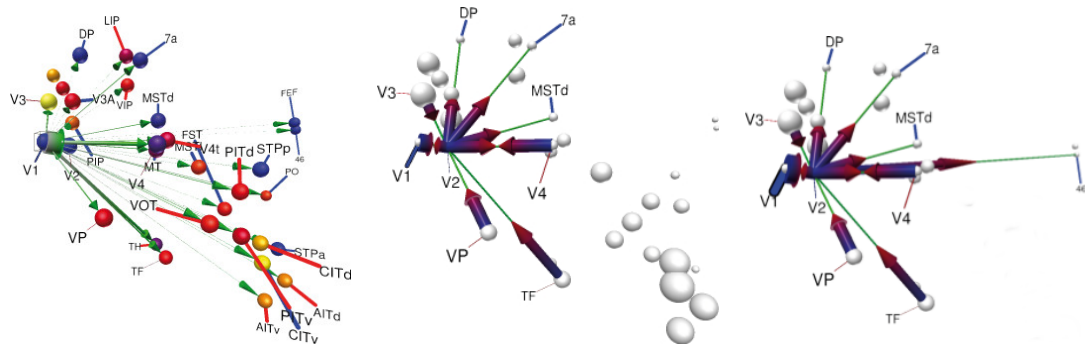


Figure 3.12: Left: Comparison of structural and functional connectivity for a simulation with transient input to V1. A strong structural connection exists from area V1 to V2. Center and Right: The *flux view* reveals a transient increase of the transmission from V1 to V2, in line with the anatomy. Fluxes are thresholded to enable focusing on the main pathways.

the first use case. Overall, the CMV system is tailored to identify contrasting interactions among areas (**RQ15**) and assess activity propagation (**RQ16**). To this end, the *Geometry Views* and *Population MFR Views* connect to the *Time Navigator* to coordinate visualization time. This ensures that displayed activity levels are in sync and allows to quickly identify interactions on different time scales (**RQ15**). Whenever an in-depth assessment of a simulation is required, the CMV system for the first use case can be started in parallel by configuring the *Simulation Data Provider* to use the desired simulation dataset.

In summary, user feedback on the CMV systems was overall positive. The semantic-aware CMVs allow domain scientists to identify areas of interest and interactively analyze neural activity data in more detail. These capabilities directly address shortcomings in previous non-interactive workflows. The CMV systems have been used to simultaneously visualize 32 vision-related areas of the macaque visual cortex where visualization designs for the presented views were developed in close collaboration with neuroscientists. Concepts of EDA were applied to integrate a variety of data modalities, e.g. geometrical data, raw simulation output, and derived information. The resulting CMV systems enable the comparison of simulations by constructing suitable event-flow networks and allow for the integration of further data modalities and visualization designs without changing existing services. Domain scientists made several discoveries in the data because the CMVs allowed them for the first time to interactively navigate and link data modalities in a readily accessible form.

3.2 Statistical Analysis of Spike Trains

3.2.1 Motivation

In this section, a CMV system for the statistical analysis of spike data is introduced based on activity data resulting from a neural network simulation.

Analysis of spike data can be roughly divided into two common approaches: statistical methods for the identification of patterns in spike data and interactive exploratory methods to facilitate humans' abilities for pattern recognition. Ideally, both approaches comple-

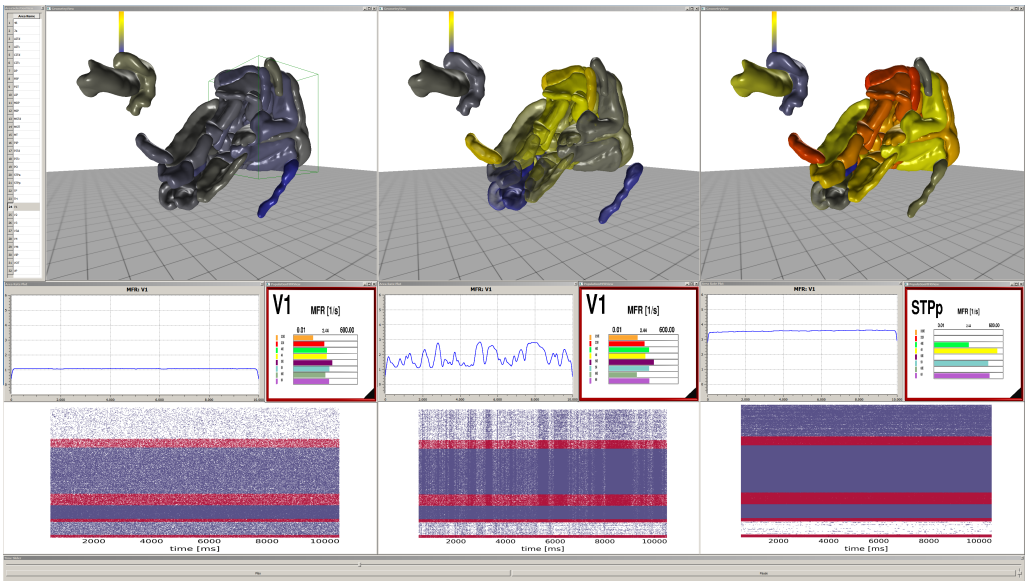


Figure 3.13: A slight alteration of the view arrangement for the comparison use case. Here, the *Time Series View* and *Population MFR View* replace the firing distribution and power spectrum. Otherwise, the view arrangement is kept consistent with Figure 3.9.

ment each other. When studying the behavior of neural systems, the inner biochemical processes of a neuron are often neglected and reduced to a *spike event*. A spike event occurs when the action potential of a neuron exceeds a predefined threshold. This process emits an electrical signal to interconnected neurons, resulting in information exchange. Spike events of a neuron over time are termed a *spike train* and a set of neurons which are organized as networks are referred to as *assemblies*. Features of interest in spike trains are irregularities in spiking behavior, synchrony of neurons, i.e., neurons spiking at the same time or within a consistent time interval, oscillations, and overall correlation patterns.

A key challenge in the visual analysis of spike train data is the multitude of spike events imposing several visualization challenges:

- the amount of spike events produce visual clutter, especially for dense spike activity where records of spike events last over longer periods than are representable in the plot.
- spike events, when plotted, might overlap because of successive firings in a short period of time.
- interaction with the data , e.g., zooming, selecting spike trains, etc. requires interactive rendering techniques. However, rendering large amounts of spike data is still a performance bottleneck for GPUs.
- the integration of statistical methods, primarily originating from experimental neuroscience, is still lacking.

To this end, the PSE is extended by views that focus on the interaction with and interactive rendering of spike trains resulting in a CMV system. This system replaces the static depiction of spike activity with interactive, coordinated views, et al., spike distributions and statistical measures such as ISI-distance. In addition, zooming and filtering, closely following the *information seeking mantra* [Shneiderman, 1996] is supported to allow for

a closer inspection of features. The integration of statistical measures facilitates a CMV system that links both analysis approaches.

3.2.2 Workflow Analysis

The investigation of spike data aims to assess the behavior of a neural system over time. While obtaining spike data fundamentally differs in computational neuroscience to experimental neuroscience, the analysis questions are similar: identification of key dynamics on a broad range of time scales resulting from stimuli, network activity, and network interaction. Scatter plots, also termed *raster plots* in this context, and histograms are classical visualization methods [Cleveland *et al.*, 1985] used for this data. However, Spence [2001] already emphasized the importance of interaction with the data to extract insight and support the decision making process. Also, Fekete & Plaisant [2002] point to the challenges of depicting multitudes of data items in a single view which concerns aspects of perception, interaction, and visualization. An exploratory tool must support users in interacting with the data: selecting, navigating, and zooming spike trains. Interaction metaphors in scatter plots are presented in Ellis *et al.* [2005] where a sampling lens is introduced to reveal hidden features in the data resulting from over-plotting caused by dense point distributions. Similar, we observe over-plotting with dense spike activity when rendering raster plots. The sampling lens uses random sampling as a density reduction technique to adjust its diameter and sampling rate.

Current research suggests that spike frequency is the principal means to carry information. Thus, spike synchronization might be driving information processing [Borisyuk & Borisyuk, 1997; Segev *et al.*, 2004], making the investigation of synchrony patterns a key feature to look for. To this end, Kreuz *et al.* [2015] introduce SPIKY, a GUI for monitoring spike synchrony for simulation and experimental data. SPIKY provides a UI for three statistical methods to detect spike train synchrony: ISI-distance [Kreuz *et al.*, 2007], SPIKE-distance [Kreuz *et al.*, 2013], and event synchronization [Quiroga *et al.*, 2002]. The ISI-distance is a parameter free, time-scale independent measure based on inter-spike intervals. The SPIKE-distance is a similar measure but can be estimated in real-time, making it applicable for interactive visualization on continuous spike train recordings. Event synchronization is a sophisticated coincidence detector which relies on simultaneous appearing spikes. SPIKY provides visualizations of three measures by plotting these next to the raster plot.

In a common analysis workflow, two spike distribution histograms often accompany the raster plot to assist in the interpretation of data: one histogram (STH) shows the spike activity distribution over time intervals (see Figure 3.14, left). The time domain is subdivided into intervals and spike activity subsequently binned. The STH is usually parameterized with a predefined bin size which can be estimated by minimizing the mean integrated squared error [Grün & Rotter, 2010] but should ideally be varied according to domain experts in an exploratory setting to discern underlying spike rates. The second histogram (PSTH) shows the summed spike counts in groupings, i.e., populations or trials (see Figure 3.14, right). While the overall identification of patterns is important, experimentalists also focus on pair-wise correlations in spike trains using more sophisticated statistical analysis measures such as the ISI-distance or event synchronization [Berger *et al.*, 2010; Grün & Rotter, 2010]. However, a severe drawback of the static depictions is the lack interaction: whenever a representation is changed, e.g., the symbol or size for

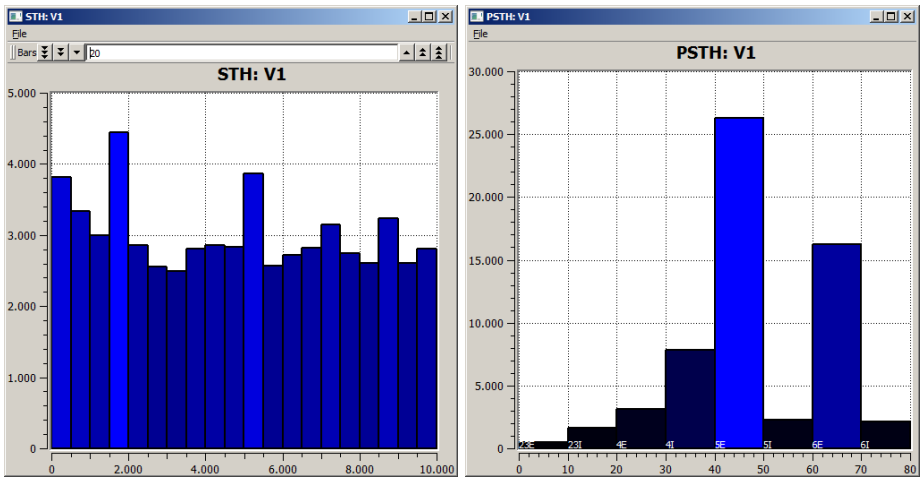


Figure 3.14: The STH (left) shows binned spike activity. The user can modify the number of bins used to slice time. Then, all spike events are plotted that fall into bins. The PSTH (right) counts all spike activity for a grouping of neurons to assess the most active and inactive groups in the data.

depicting a spike in the raster plot, a slow hand-crafted plot generation process has to be carried out. To this end, the goal for the CMV system to assist in the workflow is:

1. to interactively render raster plots, allowing to zoom, pan, and to adjust symbol size used for the representation of spike events.
2. to filter spike trains by their population or experiment to hide point distributions which are not in focus during analysis [Connors & Gutnick, 1990; Gray & McCormick, 1996; Izhikevich *et al.*, 2003].
3. to indicate over-plotting by measuring spike distance in pixel space to enhance contrast of nearby spikes.
4. to provide an interactive STH- and PSTH histogram with adjustable parameters.
5. to plot the ISI-distance for a pair of spike trains next to the raster plot.
6. to show the covariance distribution for selected ISIs.

To realize this CMV system, the following section outlines the input- and output data.

3.2.3 Input and Output Data Analysis

For this use case, the CMV operates on the simulation data presented in Section 3.1.3. Mainly, the following data is used and derived from the simulation :

1. **Spike train:** a spike train is condensed representation of a single neuron's activity over time. To condense, a target electrical activity is defined as a threshold barrier. Once exceeding the threshold, the neuron has spiked. Thus, a spike train is a time series recording denoting when a neuron has spiked. Spike trains form the input data for this CMV system.
2. **ISI-distance:** the ISI-distance [Kreuz *et al.*, 2007] is a spike synchrony measure that compares two spike trains for similar activity patterns. The ISI-distance is a measure computed by the system and visualized in a CMV.

3. **STH histogram:** the STH histogram shows the distribution of spike activity over a time period. To this end, the time domain is subdivided into equidistant intervals and spike activity is subsequently binned. The STH histogram is computed on the basis of spike trains and is output data.
4. **PSTH histogram:** the PSTH histogram shows the summed spike count of groups of neurons, i.e., populations or trials. The spike count for each group is depicted as a bar in the histogram. The PSTH diagram is based on all spike trains of neuron groupings and is output data.
5. **Covariance Distribution of ISIs:** the covariance distribution of ISI-distances depicts the covariance value for each spike train's ISI-value in a histogram. This measure is based on the two spike trains and is displayed in a CMV.

Based on the input and output data and the workflow, requirements for the CMV system are presented in the next section.

3.2.4 Requirements Analysis

To derive requirements for the CMV system, a qualitative user study was conducted using a minimalistic prototype. The prototype was projected to a big screen while a think aloud experiment [Lewis *et al.*, 1993] with 11 participants from experimental (4) and computational neuroscience (7) was performed. All neuroscientists had extensive knowledge on working with raster plots. The prototype interactively rendered raster plots and showed both STH and PSTH histograms. Neuroscientists were specifically asked to outline how they would like to interact with the raster plot. Subsequently, feedback was categorized into requirements to determine common features for experimentalists and computational neuroscientists. While experimentalists usually analyze fewer neurons over many trials, computational neuroscientists focus on a larger set of neurons in a manageable amount of populations. Computational neuroscientists suggested to focus on displaying large amounts of spike trains while experimentalists stressed that linking statistical measures is important. All participants emphasized the visual design needs to enhance contrast, to closely resemble the traditional approach for depicting raster plots, and should support intuitive interaction to improve the evaluation process. In summary, the feedback revealed the following observations resulting in requirements:

1. **Enhance contrast (R1):** enhance the contrast for all visual elements by minimizing color usage.
2. **Adjust symbol size (R2):** to change the symbol size depicting spikes helps to investigate dense raster plots. This change shall be instantaneous to not distract the interpretation task.
3. **Zoom and filter (R3):** interactive zooming and filtering the activity data is useful but needs to be responsive. While zooming the plot, visual fidelity is important.
4. **Free aspect ratio (R4):** freely changing the aspect ratio of a raster plot helps to spot synchrony patterns.
5. **Compare firing rates in specific time intervals (R5):** comparing firing rates in adjustable time intervals is useful. Allow to interactively parameterize time intervals.

6. **Compare firing rates between groups (R6):** to compare firing rates between groups of neurons is important. Allow to select groups and show comparison measures.
7. **Link the data (R7):** link interactions of different data representations to easily orient in the dataset. A key challenge in the analysis is the vast toolbox of statistical methods and rapidly changing questions about the data. To verify hypotheses, the ability to show statistical measures next to the raster plot is beneficial. The semantic-aware architecture provides flexibility to rapidly integrate statistical methods.
8. **Use the raster plot for selections (R8):** use the raster plot to select spike trains for statistical analysis.
9. **Interactively display large amount of spike trains (R9):** when interacting with large spike activity data, rendering performance is critical. When selecting spike trains, visual feedback should be provided.
10. **Avoid visual clutter (R10):** the depiction of large amounts of spike activity is prone to visual clutter. Indicate over-plotting in dense spike trains to avoid misleading results and interpretation mistakes.
11. **Stream spike activity from running simulations (R11):** when streaming activity data directly from a running stimulation, a dynamic mapping technique for spike trains is required because the plot's dimension to show all data are unknown. Also, rendering performance is critical.
12. **Integrate analysis techniques (R12):** leverage statistical analysis techniques already developed in the neuroscience domain.

Based on these requirements, a CMV system is developed which mimics the original static depictions. It adds interaction capabilities, utilizes interactive rendering, and offers statistical analysis techniques by integrating the Elephant [Berger *et al.*, 2010]. In the following section, the resulting views will be presented.

3.2.5 Views and Services

The visual design derived from the prototype consists of eight coordinated views (**R7**): the *Raster Plot View*, the *Time Series View*, the *STH-* and *PSTH View*, the *Covariance Distribution View*, the *Spike Train Comparison View*, the *Spike Train Selection View*, and finally the *Area Selection View* (see Figure 3.15). In the following, each view is presented in more detail:

Raster Plot View — *Raster plots* or *rastergrams* are a specialization of scatter plots commonly used to represent spike activity to assess (see "overview first" [Shneiderman, 1996]) a neural system's behavior (see Table 3.1, **D1**) over a recording period. In raster plots, the x-axis depicts time while on the y-axis a spike train is plotted. For each spike of a neuron, a symbol is drawn along the x-axis as seen in Figure 3.16. The *Raster Plot View* allows to zoom and pan to closer inspect spiking activity (**R3**), and to perform selections of spike trains via a pointing device (**R8**) which are subsequently used in the statistical views, e.g., the *Covariance Distribution View* (**R7**). As a simple means to encounter visual clutter, the symbol size of a spike is interactively adjustable (**R2**) by clicking on the

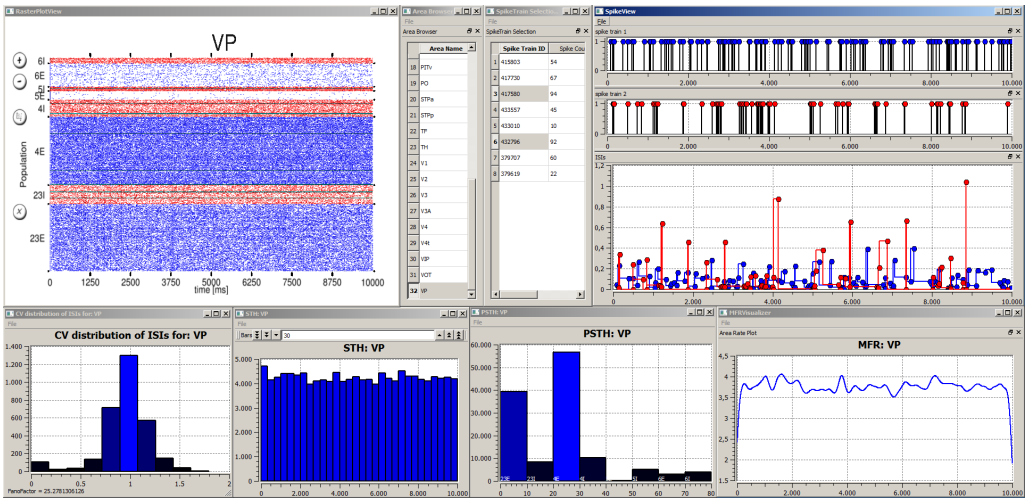


Figure 3.15: The *Raster Plot View* (top, left) shows spiking activity for a brain area selected in the *Area Selection View* (top, right next). In the top middle, the *Spike Train Selection View* displays the spike trains from the *Raster Plot View*. Here, two spike trains can be selected for the *Spike Train Comparison View* to inspect synchrony. In the bottom left corner, the covariance distribution of all selected spike trains is shown, followed by a STH and PSTH diagram for the brain area. In the bottom right corner, the mean firing rate for the area is plotted.

$+$ and $-$ buttons as shown in Figure 3.16 (left). Depending on whether a neuron is considered inhibitory or excitatory, it is respectively grouped into populations. To minimize color usage and enhance contrast (**R1**), inhibitory populations are colored in red while excitatory ones are depicted in blue. Moreover, the raster plot's aspect ratio is freely adjustable by stretching or squeezing (**R4**). This makes the assessment of synchrony easier as vertical stripes will collapse to occupy less pixel space. The *Raster Plot View* uses a tile-based off-screen rendering approach, exploiting that spike activity can only advance in time. This enables the interactive plotting of spike activity from a running simulation (**R9, R11**) via a spike activity data *slot*. To this end, spike activity is sliced into time intervals and mapped to tiles. Subsequently, tiles are stitched together in a framebuffer object. Whenever new spikes are rendered, they are either inserted into an existing tile or a new tile is added. This way, plotting the entire spike activity is reduced to updating a tile, drastically improving rendering performance. In addition, storing spike activity directly on the GPU allows for the interactive adjustment of symbol size (**R2**) or recoloring spike trains (**R1**). This rendering technique stores spike activity in a vertex buffer. A distance function d and an energy term e is defined to detect and indicate over-plotting caused by dense spike activity where distinct spike events are mapped to the same pixel-space (**R9, R10**):

$$d_{id,r}(s,\{x_0,\ldots,x_n\}) = max(0, 1 - \sum_{i=0}^{n} 1 - \frac{min(|s-x_i|, r)}{r}) \tag{3.5}$$

$$e(id,s) = \sum_{i=0}^{n} max(|s - spike_i|, 1) \tag{3.6}$$

where id is the spike train, n its number of spikes, s the current position in time, $spike_i$ the point in time when a spike occurs, and r the radius of the spike's depiction symbol.

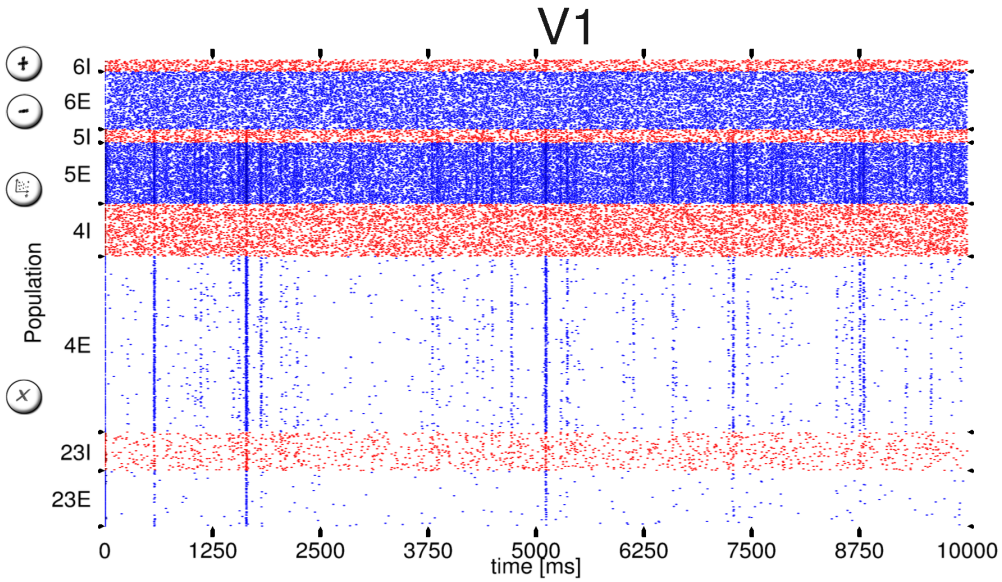


Figure 3.16: A raster plot showing spike activity: the x-axis depicts time while on the y-axis neurons are assigned. For each spike of a neuron, a filled circle indicates a spike event. Red circles show inhibitory neurons while blue ones depict excitatory ones. Vertical stripes show synchrony in spike trains.

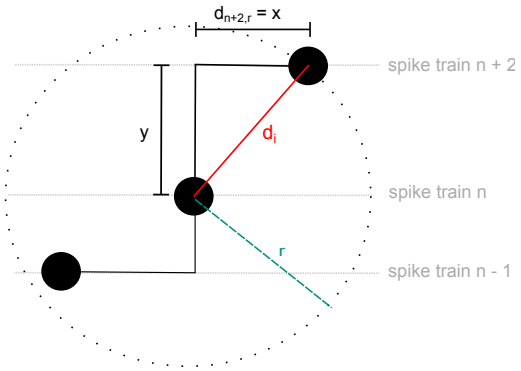


Figure 3.17: Illustration of the process to determine d_{min}.

While rendering, the distance function and energy term are evaluated and stored in a two-channel texture. Both functions are accumulated whenever new spikes are rendered. To render spike trains, for each tile a quadrilateral is drawn which emits fragments. For each fragment, a lookup in the distance field is performed which sweeps around the current spike along the radius r to retrieve nearby spike distance values j. Then, the fragment distance d_i at its position i to all nearby spikes is calculated by iterating over each distance value around r where the distance d_i is given by the Pythagorean theorem. Then, the minimum d_{min} of $D = \{d_0, ..., d_j\}$ is obtained as illustrated in Figure 3.17. If $|d_{min}| \leq r$, a fragment for depicting the spike event is generated, otherwise it is discarded. The color of the fragment is determined by multiplying the user-defined color for a spike with a factor l, where l is:

$$l = \frac{1 - d_{min}}{e_i} \tag{3.7}$$

and e_i is the energy from Equation 3.6 at the position i in the distance field. The fragment's

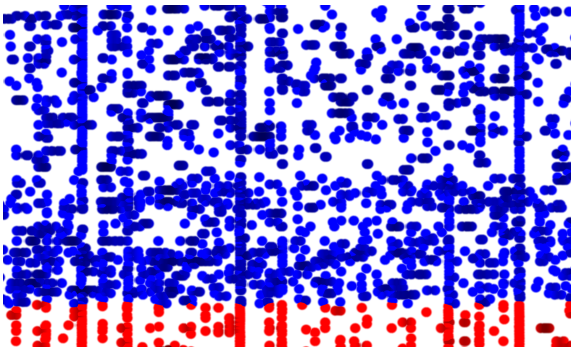


Figure 3.18: Over-plotting is indicated by a slight decrease in luminance which helps in spotting dense spike activity.

α- value is determined by Hermite interpolation s:

$$\alpha = s(r-\varepsilon, r+\varepsilon, d_i) \tag{3.8}$$

where ε is the gradient magnitude of two neighboring texels at the fragment's position in the quadrilateral. Finally, the fragment is blended with the α- value to indicate over-plotting. This results in a color with darker luminance for spikes in close proximity, indicating over-plotting as seen in Figure 3.18.

STH View — The *STH View* depicts the distribution of spike activity over time to assess firing behavior (**R5, R6**). It supports zooming and panning while also interactively changing the numbers of bins. This allows to reveal details over the activity distribution. The *STH View* can either be used to plot the activity distribution of the entire population data or can be coordinated with the spike trains selected from the *Raster Plot View* to facilitate **R6**. Additionally, a color table maps the number of spikes for each bin to a color to ease the perception of change w.r.t. the height of a bin (cf. Figure 3.14 (a) and (b)).

PSTH View — The *PSTH View* depicts the firing behavior of populations in a histogram for comparison tasks (**R6**). The histogram displays spikes for all neurons in a population as a bar where the bar's height encodes the total number of spikes. An example of this view is shown in Figure 3.14 (b).

Covariance Distribution View — The *Covariance Distribution View* operates on a selection of spike trains and calculates their ISI-values which are subsequently used to compute their covariance. The covariance is then displayed in a histogram (see Figure 3.15 first view, second row). In addition, it calculates the Fano Factor [Kamil Rajdl, 2014] (**R12**) to describe variability in both spike trains and displays it in the bottom left corner. The computation of ISI-values and the Fano Factor are performed by the Elephant toolkit (**R12**). Spike selection is coordinated from the *Raster Plot View* (**R7, R8**)

Spike Train Selection View — The *Spike Train Selection View* displays a list of neuron IDs selected from different views, e.g., the *Raster Plot View* (**R7**). It acts as a clipboard (**R8**) for spike train selections (**R8**) and shows total number of spikes next to the neuron ID. It is used in conjunction with the *Spike Train Comparison View* to assess synchrony in pairs of spike trains (**R7**). Figure 3.15 (third view, first row) shows a set of spike trains that were selected in the *Raster Plot View*. The user can now filter and select a pair that is passed over to the *Spike Train Comparison View*.

Spike Train Comparison View — The *Spike Train Comparison View* enables the assessment of synchrony between two spike trains. To this end, it shows a pair of spike trains to assess synchrony, i.e., correlation of spike events (**R5**). It receives a pair of spike trains (**R7**) from the *Spike Train Selection View* and displays spike timings vertically aligned as shown in Figure 3.15 (last view, first row). The view also calculates and plots the ISI-distance for both spike trains using the Elephant toolkit (**R12**).

In summary, the *Area Selection View* is used to select a brain area of interest which is coordinated with the *Raster Plot View*, the *STH View*, the *PSTH View*, and the *Covariance Distribution View*. The *Spike Train Selection View* displays all selected spike trains from within the *Raster Plot View* and allows to subselect a pair of spike trains which is subsequently passed to the *Spike Train Comparison View* for comparisons tasks of spike train synchrony. The next section outlines the event-flow network to construct this CMV system.

3.2.6 CMV System Construction

Key research questions to analyze spike trains resulting from a simulation of the multi area model presented in Section 3.1 are:

- **RQ1:** are there any synchronous activity patterns visible in the data?
- **RQ2:** are there similar activity patterns visible in different brain areas?
- **RQ3:** is there statistical significant synchrony between pairs of spike trains?
- **RQ4:** how is the spike activity distributed in a brain area?
- **RQ5:** how is the covariance of ISIs distributed between selections of spike trains?
- **RQ6:** how is firing behavior of populations in an area distributed?
- **RQ7:** is there activity in areas and how does the activity "look like"?

To assess these questions, the PSE is constructed using the following services to form a CMV system:

1. the *Simulation Data Provider* to access the simulation datasets, i.e., spike data.
2. the *Raster Plot View* displays spike data for a brain area. It also allows to select spike trains subsequently used in other views (**RQ1, RQ2, RQ6, RQ7**).
3. the *Area Selection Service* allows the selection of areas.
4. the *Time Series View* shows the activity data (MFR) (**RQ7**).
5. the *STH View* displays the spike activity distribution (**RQ4, RQ6**).
6. the *PSTH View* allows the assessment of spike activity between populations (**RQ4, RQ6**).
7. the *Covariance Distribution View* shows the covariance distribution of a selection of spike train ISIs (**RQ5**).
8. the *Spike Train Selection View* is used to select a pair of spike trains (**RQ3**) from the *Raster Plot View*.

9. the *Spike Train Comparison View* enables the assessment of synchrony between two spike trains (**RQ1, RQ3**).

Based on these services, the event-flow network is constructed to form a CMV system as shown in Figure 3.19. The event-flow network and all used services are also stored in the persistent ontology to start the CMV system directly from within the *runtime system*. In

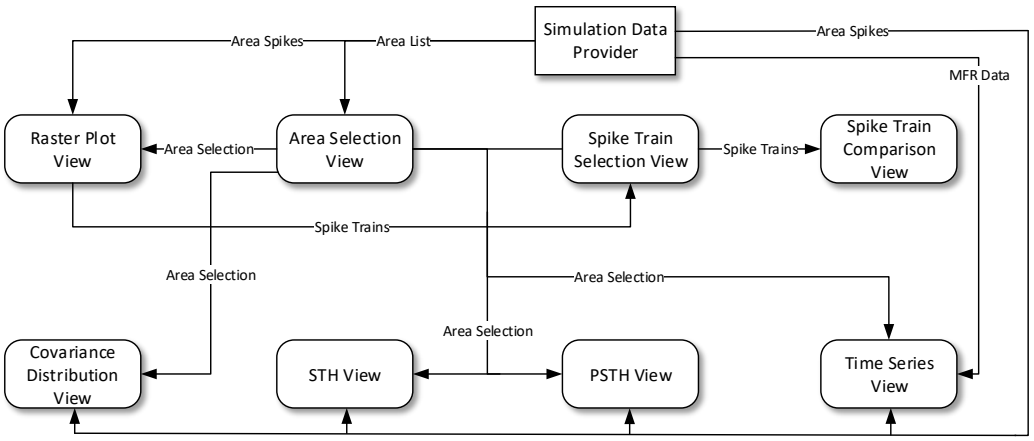


Figure 3.19: The event-flow network resulting from the coordination of services: the *Simulation Data Provider* offers access to the spike activity data, list of brain areas, and mean firing rates. The *Raster Plot View* receives spike data and displays the plot depending on the selection in the *Area Selection View*. The *Time Series View*, the *STH View*, and *PSTH View* is updated whenever a selection is performed. The *Covariance Distribution View* and *Spike Train Selection View* is coordinated with selections of spike trains from the *Raster Plot View*. When a pair of spike trains is selected in the *Spike Train Selection View*, the *Spike Train Comparison View* is updated to display the two spike trains.

the following section, this CMV system is used to assess activity data.

3.2.7 Results

The *Area Selection View* forms the starting point of the investigation. It lists all brain areas from the simulation to select an area of interest which is subsequently shown in the *Raster Plot View* (**R7**). The *Raster Plot View* interactively renders a large amount of spike data (**R9**) to assess the data (**RQ7**). Users pointed out that the view's aspect ratio should not be fixed because synchronous activity appears as vertical stripes (**R4**). By stretching and squeezing the view, users have control over how much pixel space stripes occupy, making synchrony easier to spot (**RQ1**). Also, the view allows to adjust the symbol size of spikes to reduce visual clutter (**R2**) and indicates over-plotting by coloring close proximity spikes darker (**R10**). A tile-based rendering technique allows for plotting streamed spike data from running simulations (**R11**). Due to the view's interactive rendering capabilities, responsive zooming and filtering is supported without aliasing artifacts (**R3**). Contrast is enhanced by using only two distinct colors for inhibitory and exhibitory populations (**R1**). Spike trains can be selected in the *Raster Plot View* and are then displayed in the *Spike Train Selection View* which shows the selected spike train ID and its number of spikes.

Here, the user subselects pairs of spike trains for statistical analysis (**R8, R12**) in the *Spike Train Comparison View* to investigate synchrony (**RQ3**). Pair-selections can be changed interactively triggering an update of the *Spike Train Comparison View*.

To compare firing rates in time intervals (**R5**) and between groups (**R6**), both the *STH-View* and *PSTH View* are used to assess **RQ4** and **RQ6**. The *Covariance Distribution View* (**RQ5**) and *Spike Train Comparison View* (**RQ3**) integrate statistical analysis techniques from the Elephant[3] toolkit. Thus, the semantic-aware CMV system allows to construct and coordinate views utilizing analysis methods provided by community tools (**R12**).

To detect similar activity patterns among brain areas (**RQ2**), the user can switch between different raster plots by selecting areas of interest in the *Area Selection View*. In addition, the CMV system can be configured to show additional *Raster Plot Views* to isochronal assess plots. Also, the *Time Series View* displays the mean firing rate for areas selected in the *Area Selection View* to assess overall activity levels.

In summary, the CMV system is tailored to assess spike data resulting from either simulations or experiments. It presents a solution for the technical challenge of rendering large amounts of spike data while also avoiding visual clutter. In addition, it allows for interacting with the data and integrates statistical analysis techniques commonly applied in the neuroscience community.

3.3 Visualizing and Steering Structural Plasticity Simulations

3.3.1 Motivation

This use case presents a CMV system to visualize, steer, and explore structural plasticity simulations in NEST by extending the PSE. This use case is based on [Nowke *et al.*, 2018].

Structural plasticity models the dynamic creation and deletion of synapses in a neural network. However, modeling plasticity contains many degrees of freedom and interactions among elements change over time. Also, biological data to specify a complete set of parameters for such a model is incomplete and even if obtained, finding free parameters for a simulation leading to results matching experimental data is NP hard [Cubitt *et al.*, 2012]. To this end, interactive visual exploration of the parameter space through simulations to discover the system's characteristic behaviors limits the search space [Sedlmair *et al.*, 2014]. Combined with interactive simulation steering, the time for obtaining potentially "optimal" parameter space solutions is significantly reduced [Matkovic *et al.*, 2008b; Matković *et al.*, 2014]. Roberts [2007] shows that the application of the CMV paradigm supports exploratory data analysis through interactions with the data while emphasizing different details. In addition, Ryu *et al.* [2003] present CMV systems that have been successfully utilized to uncover complex relationships by relating data modalities and scales which assists researchers in context switches, comparative tasks, and supplementary analysis techniques. Further CMV systems are presented by North & Shneiderman [2000], Boukhelifa & Rodgers [2003], and Weaver [2004].

[3] `http://neuralensemble.org/elephant/`

The here presented CMV system focuses on finding suitable connectivity configurations which reflects a complex parameter search problem. The generation of local connectivity uses structural plasticity in NEST [Bos *et al.*, 2015] following homeostatic rules introduced in Butz & van Ooyen [2013]. The CMV system allows to steer structural plasticity parameters during the simulation and interactively plots firing rates and connectivity properties of populations.

The CMV system is applied to two use cases:

1. the first use case explores the generation of connectivity in a simple two population network where connectivity is tuned to a desired level of average activity by taking multiple trajectories with different biological significance.
2. the second use case considers a whole brain simulation [Deco *et al.*, 2013] where the exploration of non-unique connectivity solutions is explored.

Overall, observing the evolution of connectivity, especially where several biologically paths lead to equivalent solutions, helps to form a deeper understanding of the brain's development, learning, and repair capabilities. The CMV system provides a basis for developing new analytic and computational solutions to explore structural plasticity in neuronal and neural mass networks and promotes rigorous analysis of complex network models.

3.3.2 Workflow Analysis

Structural plasticity can be seen as a biological controller in a multi-objective optimization problem. This controlling mechanism gradually creates and destroys connections between neurons, or modifies the strength of existing synapses. Here, control signals refer to the variations in the connectivity while states refer to the dynamics of the network. The control history which satisfies the constraints of the system, i.e., experimental parameters of neurons and synapses, during a time interval is called an "admissible control" while an "admissible trajectory" is a state trajectory which satisfies the constraints of all state variables through the same time interval. Following, the final state of the system is required to be in a specific region. By applying a control signal, the system evolves from its initial state to a final state following some trajectory. The performance of this trajectory is the difference between a desired and the obtained measure for a heuristic involving the dynamics of the system. In this use case, the performance function is given by homeostatic rules. To reach a defined target activity regime, it is unknown whether an optimal admissible control exists which sends the system through an admissible trajectory for a given performance function. This optimization problem seeks a global minimum for one or more admissible trajectories of the system. Since finding the exact control signals or parameters for a simulation replicating experimental results is unachievable in polynomial time, it may still be possible to confirm solutions in polynomial time.

The primary purpose of the workflow is to study the morphological transformations neurons undergo in time leading to the creation and deletion of synapses. A secondary purpose is the automatic generation of neuron to neuron synapses models to compensate for gaps in experimental connectivity data as described in Diaz Pier *et al.* [2016]. Here, a network autonomously generates synapses to achieve a stable profile of activity by progressively changing the connections and weights between neurons to find stable config-

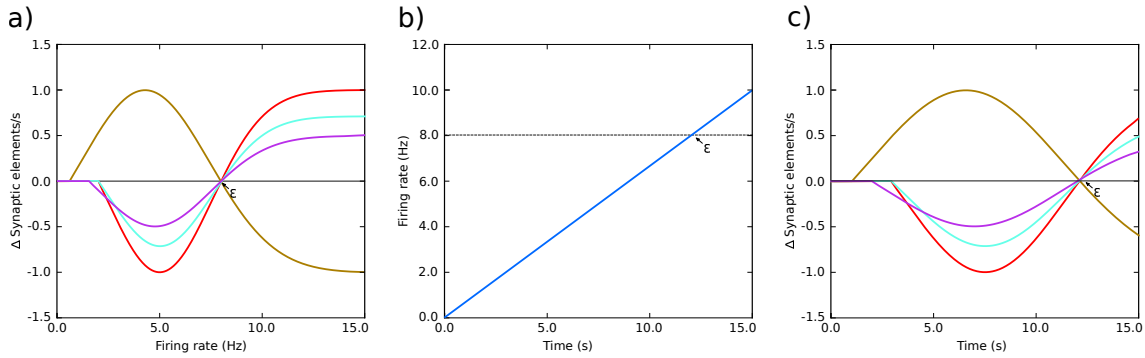


Figure 3.20: (a) Example of growth rate curves determining the creation or deletion of synaptic elements. The parameters defining the shape of the curve are the minimal firing rate η for creating or deleting synaptic elements, the target firing rate ε, and the growth rate ν which is the value of the curve in synaptic elements/s. The red, cyan and purple curves have a negative value of ν which implies that synaptic elements are deleted when the current firing rate is less than the target rate. Conversely, synaptic elements will be created when the firing rate exceeds the target. The brown curve has a positive ν which works in the opposite way. All curves display different values of η and have a target firing rate ε of 8 Hz. (b) Firing rate imposed on sample systems with Gaussian growth curves shown in (a); and c) the resulting evolution of synaptic growth rate through time due to the firing rate changes depicted in (b).

urations. Neurons have synaptic elements formed of contact points which increase or decrease in number according to homeostatic rules and are used to create new synapses. If a contact point is eliminated, the synapses is destroyed. These homeostatic rules control synaptic elements to steer the mean electrical activity to a desired state. Gaussian curves as shown in Figure 3.20 are examples of homeostatic rules defining the growth rates of connection points. The parameters defining the growth and decay of synapses are the minimum firing rate η required to generate synaptic elements, the value ν of the growth rate curve when the firing rate is $(\varepsilon - \eta)/2$, and the target firing rate ε. Modifying these values alters connectivity in the network.

Using structural plasticity for a single population to reach a targeted activity level is fast and insensitive to the choice of the parameters ν and η. However, challenges arise when structural plasticity is involved on multiple interconnected populations with differing activity levels because small changes in connectivity impact the activity of all populations leading to a propagated destabilization. In addition, the update interval at which synapses are modified has significant impact on stability. The delay between a control change and the system's response determines the capability of keeping the system stable.

In a previous workflow described in Diaz Pier *et al.* [2016], no simulation steering was possible. Due to many control parameters and observables, brute-force parameter search was insufficient to obtain stable systems. To this end, modifying the update interval and the growth behavior (ν and η) during a simulation is crucial for finding a stable state for multi-population networks. Thus, the parameter search workflow requires:

1. **Observe variables (W1):** the simultaneous analysis of several changing variables by an expert.

2. **Compare activity (W2):** comparing the activity level of several populations simultaneously.
3. **Change parameters (W3):** changing simulation parameters in each population while the simulation is running.
4. **Store connectivity (W4):** storing the connectivity state at a specific point in time.
5. **Load connectivity (W5):** loading a stored connectivity state.

This workflow requires a CMV system that enables the exploration and steering of plasticity simulations while displaying population connectivity and activity.

3.3.3 Requirements Analysis

While steering the plasticity simulation, navigating and visualizing the parameter space is required to rapidly reach stable configurations of connectivity in tightly connected populations. To this end, the following requirements were established in close collaboration with domain experts:

1. **Display time series data (R1):** display $2 \times N$ time series for (inhibitory and exhibitory) activity and connectivity where N corresponds to populations in the simulation.
2. **Plot firing rate (R2):** plot the firing rate of populations.
3. **Plot connection count (R3):** plot the number of connections of populations.
4. **Select multiple populations (R4):** select multiple populations.
5. **Avoid clutter (R5):** avoid visual clutter to distinguish populations.
6. **Control ν (R6):** control each population's growth rate ν.
7. **Control η (R7):** control each population's minimum electrical activity η.
8. **Control the update interval (R8):** control the update interval.
9. **Control the simulation loop (R9):** allow to start or stop the simulation.
10. **Load and save the network state (R10):** load and save the network state, i.e., connections and control parameters.
11. **Show multiple plots (R11):** show multiple population plots in separate views for comparison tasks.

Requirements **R1-R5** target the parameter search workflow **W1** and **W2** while **R6-R11** target **W3-W5**.

The development of the tool was organized into four stages:

1. the initial simulation script was modified to retrieve electrical activity and connectivity values while the simulation is performed.
2. the views and UI were developed.
3. parameter synchronization between the UI and the simulation was added.

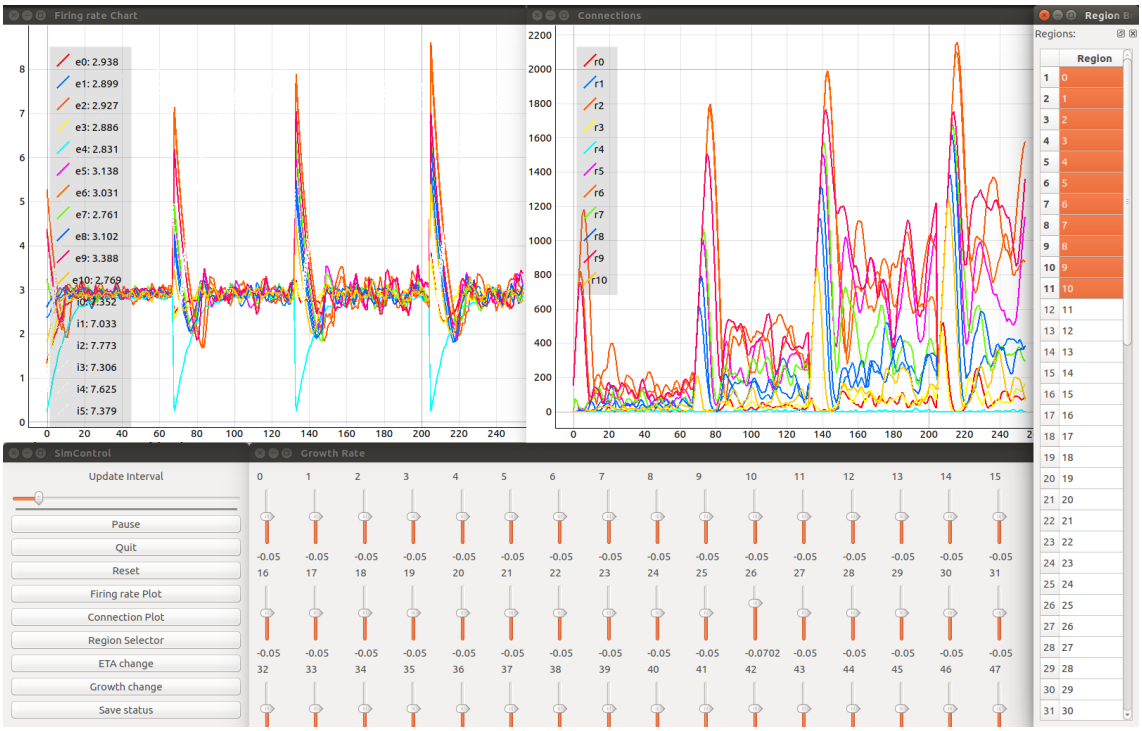


Figure 3.21: The firing rate of brain regions (upper left) and total connections (upper right) is displayed while a simulation is performed. The *Control View* and *Growth Rate Manipulation View* (bottom left and center) allow for parameter space exploration. The *Region Selection View* (far right) allows to filter regions depicted in the plots.

4. the simulation script was optimized to run on supercomputers.

Steering the simulation requires bidirectional communication between the CMV system and the simulator. For each simulation step, all firing rates of populations and total connections are obtained from the simulation. Then, control parameters, i.e., η, ν of each population, the update interval, and basic commands are sent to the simulator and applied in the next simulation loop iteration. Bidirectional communication is realized via the slot communication provided by the EDA.

The next section presents the developed views for the CMV system.

3.3.4 Views and Services

The CMV system consists of six services implementing requirements **R1-R11** as can be seen in Figure 3.21:

Control View — The *Control View* allows to start the simulation, the steering interfaces, the *Color Editor View* and the *Region Selection View*. To control the update interval (**R8**) and to pause or restart the simulation (**R9**) is also performed here. To reuse connectivity in neighboring points of the parameter space, a save- and load functionality is provided (**R10**). To this end, each population's η and ν values, the update interval, and connections between neurons are saved. To reuse a snapshot, synaptic elements with their associated growth curves are reconstructed using the stored η and ν values. Then, synaptic elements are registered in the structural plasticity framework and the update interval is set. Finally,

Figure 3.22: The *Color Editor* enables the customization of the firing rate and connectivity curves in the plot. Here, a color for a region's inhibitory (I) and excitatory (E) population can be selected and line drawing style as well as thickness can be controlled.

all connections are recreated. In this way, a network with differing global parameters can start from a partial solution and arrive at the target activity more quickly.

Region Selection View — The *Region Selection View* lists all regions (see Figure 3.21, rightmost). It allows to select multiple regions, coordinated with all other views, to plot their activity and connectivity (**R4**). Individual regions can be investigated by double clicking on a region: this opens a standalone *Activity Plot View* and *Connection Plot View*, useful for pairwise comparison tasks (**R11**).

Activity Plot View — The assessment of simulation results is based on the population activities. To this end, the *Activity Plot View* plots firing rates of populations (**R1**) selected in the *Region Selection View* (**R4**) and visualizes the trajectories of the network. The labels *e* and *i* in the plot identify excitatory- and inhibitory populations. To retrieve the firing rates (**R2**), it connects to the simulation. Interactive zooming and panning allow to focus on details on demand facilitating the "information seeking" mantra [Shneiderman, 1996]. Axes can be independently scaled and the data range confined. The view also exports the plot as a figure. To distinguish curves (**R5**) associated with a population, a color table is used, defined via the *Color Editor View* in addition to numbers identifying regions. Multiple *Activity Plot Views* can be opened in the *Region Selection View* (**R11**).

Connectivity Plot View — The *Connectivity Plot View* (see Figure 3.21, upper right) displays the total number of connections in a population (**R3**). Because structural plasticity changes a population's connections depending on its firing rate, the view helps to verify the structural plasticity model and shows the trajectories of the network. The label *r*, displayed in the legend in the top left corner, and the population's number identify the connectivity values. The view is also coordinated with the *Region Selection View* (**R4**) and the *Color Editor View* (**R5**). Like the *Activity Plot View*, it offers interactive zooming and panning. Likewise, axes are automatically scaled such that all retrieved connectivity values are depicted. The plots can also be exported as figures for the tracking of results or publication purposes.

Color Editor View — The *Color Editor View* presents a GUI to define a color, the line drawing style, and line thickness to distinguishably plot a population's activity and

connectivity curve (**R5**). A color for a population's inhibitory- and excitatory population can be defined by clicking on the corresponding list entry. The *Activity Plot View* and *Connectivity Plot View* are coordinated to synchronize its settings. The settings are also saved to disk for later use.

ETA- and Growth Rate Manipulation View — Both, the *ETA Manipulation View* and *Growth Rate Manipulation View* present GUIs to change η and ν parameters while the simulation is running. They allow to conduct a parameter space exploration (**R6** and **R7**) by steering the control signals to take the system from its current state to a desired final state. Both views are started within the *Control View*. Each view presents a slider for each population to control either η or ν, respectively. The slider is named after the population and shows the currently applied value. Whenever its value is changed, it is send to the simulation and subsequently processed in the next simulation loop iteration.

In the next section, the construction of the event-flow network for this CMV system is presented.

3.3.5 CMV System Construction

The CMV system is constructed to investigate the the following key research questions:

- **RQ1:** what trajectories lead to solutions?
- **RQ2:** which solutions are biologically meaningful?
- **RQ3:** how do solutions compare to each other?
- **RQ4:** what are the implications of different parameter setups in a network model?
- **RQ5:** can the model be validated?
- **RQ6:** can biologically meaningful populations for the simulation be defined?
- **RQ7:** can measures for future automatic or semi-automatic assessments of model behavior be derived?

To allow for the assessment of these questions, the CMV system's event-flow network is established among the following views:

1. the *Region Selection View* is used to select brain regions.
2. the *Activity Plot View* to display activity levels.
3. the *Connectivity Plot View* to show the number of connections of selected regions.
4. the *Color Editor View* to define the coloring and line properties in the the *Activity Plot View* and *Connectivity Plot View*.
5. the *ETA Manipulation View* to change electrical activity for regions to influence the creation and destruction of connections.
6. the *Growth Rate Manipulation View* to define the rate of growth for each region's plasticity.
7. the *Simulation* which acts as a *data provider* to views and applies user interaction.

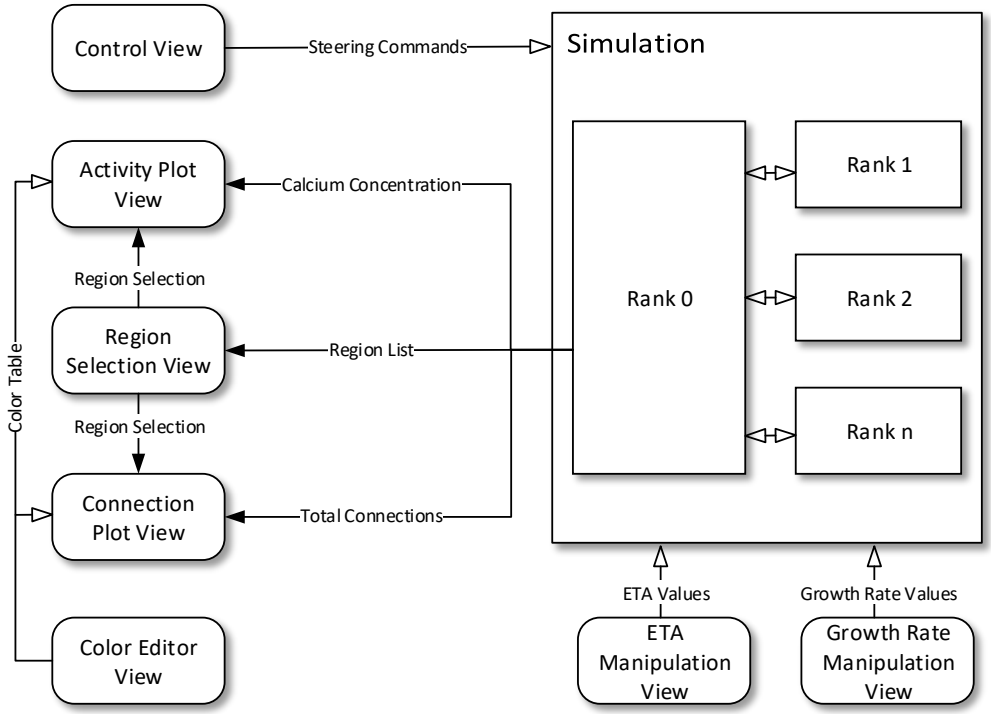


Figure 3.23: Overview of the system architecture: boxes denote individual services. Black arrows mark communication from the simulator to views. White arrows indicate event-flow from views to the simulator. Ranks indicate MPI processes responsible for the parallel computation of the neural network.

The resulting event-flow network is depicted in Figure 3.23. The simulation wraps MPI-processes to compute the neural network in parallel. The rank 0 MPI-process has a steering *slot* to receive commands from views and gathers data from all other ranks. Finally, it forwards gathered data to views.

In the next section, results obtained using this CMV system are presented.

3.3.6 Results

The CMV system was used to evaluate and obtain results for two use cases. For the first use case, structural plasticity simulations before the CMV system was developed were reported in Diaz Pier *et al.* [2016]. Due to the large number of curves, the inability to focus on populations, and the lack of interactivity, static visualization made it very difficult to identify the evolution of connectivity in relation to parameter changes. Moreover, a new simulation run was required whenever any parameter is changed. Even when some populations had reached their target activity, it was challenging to identify suitable trajectories that lead to stable solutions for all populations.

The simulated system is constrained by connectivity data and activity levels obtained from experimental measurements but the constraints still allow to reach non-physiological states. The system may have several trajectories to reach these states, indicating that the system is under-constrained. However, there are many admissible trajectories which take

the system to biologically plausible states. At first glance, it is unclear how to explore the parameter space as many parameters and long computation times make it unfeasible to find stable populations with a brute force approach and no heuristic is available to reduce the dimensionality. Without expert knowledge in a closed loop setup, admissible trajectories are hard to find.

In the first use case, a model with two point neuron populations, one excitatory and one inhibitory, is used. The network contains 1000 leaky integrate-and-fire neurons with exponential shaped post-synaptic currents of which 80% belong to the excitatory population and the rest to the inhibitory one. All neurons receive independent background excitatory Poisson noise with a rate of 10 kHz. At the beginning of the simulation, no connections between neurons exist. The system is allowed to create both excitatory and inhibitory connections through structural plasticity. The evolution of the firing rate and the growth of connections is regulated by two homeostatic rules defined by Gaussian curves. The target average activity of the inhibitory population is set to 20 Hz while the target average activity in the excitatory population is 5 Hz. The evolution of connectivity is guided by modifying the growth rate and the Gaussian curve.

The CMV system assists to observe the path to a solution (**RQ1**). It allows to assess how a faster growth of inhibition triggers an overshoot in the generation of excitatory connections which results in a rewiring of the system. By regulating the speed of creating connections, the CMV system helps to explore different paths to a solution where the relationship between excitation and inhibition changes over time. Figure 3.24 shows the evolution of growth rate, firing rate, and connectivity for six examples of multiple trajectories and connectivity. All examples start with a growth rate of 0.0001 synaptic elements / ms. Figure 3.24a shows a smooth growth where the control signals were modified by reducing the growth rate to 0.00005 at iteration 8 (mark a.1) to regulate the overshoot in the inhibitory population. Figure 3.24b shows a simulation where the control signals for growth start very fast, producing a constant oscillatory behavior. This is achieved by changing the growth rate from 0.0001 to 0.001 at iteration 38 (mark b.1) and then to 0.003 at iteration 80 (mark b.2). Following, the update interval in connectivity is changed to 500 which produces a big oscillation triggering a rewiring of the network (mark b.3). Finally, the growth rate is set to 0.00005 at update 161 (mark b.4) to settle the system at a stable state. Figure 3.24c illustrates a very fast growth at the beginning by changing the growth to 0.004 at iteration 46 (mark c.1). Then, a sharp change to slow growth when the system is oscillating close to the target firing rate. The growth rate is changed to 0.0018 at iteration 98 and further down to 0.0007 at iteration 103 (marks c.2 and c.3). Figure 3.24d shows a case which seems stable in terms of activity but is unstable in terms of connectivity as it exhibits a constant race between excitation and inhibition to maintain the target activity. Growth rate is set to 0.001, at iteration 24 (mark d.1) to 0.0056, at iteration 52 (mark d.2) and to 0.0020 at iteration 78 (mark d.3). Figure 3.24e shows a trajectory which is not biologically meaningful, as we have built a network with only excitatory connections by modulating the growth of connections very carefully around the target activity. Here, a growth curve is defined that does not allow for the creation of inhibitory connections unless the activity is above a desired firing rate. Finally, in Figure 3.24f, a trajectory is shown which is not admissible in terms of biological meaningfulness because the network is taken to very high firing rates before it settles back to its target. At iteration 17, the growth rate is set to 0.002 (mark f.1) and then slowly down to 0.00056 at iteration 60, to 0.0002 at iteration 80 and finally to 0.00005 at iteration 90 (marks f.2, f.3 and f.4). This

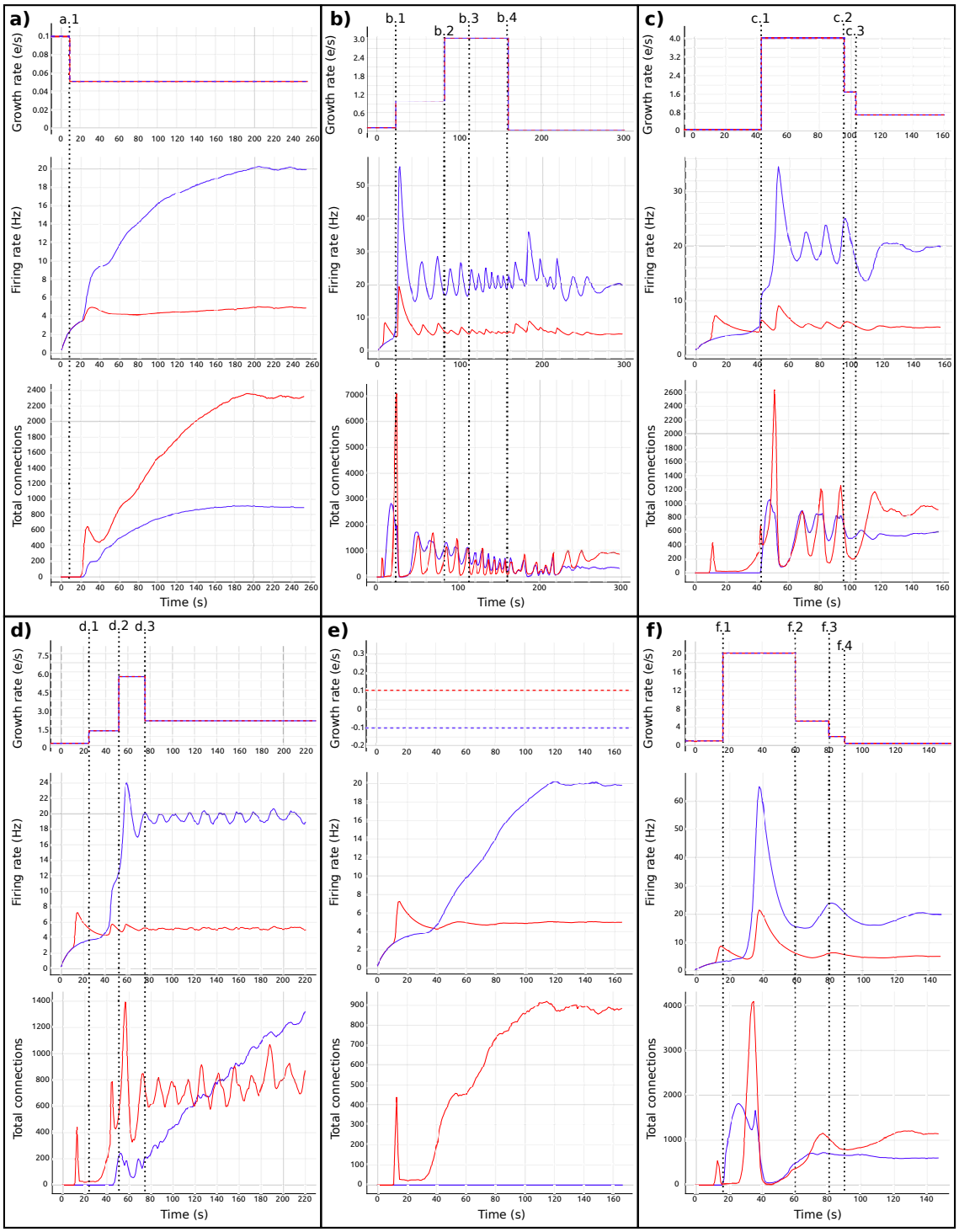


Figure 3.24: Evolution of growth rate (top), firing rate (middle) and outgoing connections (bottom) for six different trajectories (a-f) for the two population model. Excitatory neurons are marked in red and inhibitory neurons are blue. Vertical dashed lines correspond to changes to the growth rate (top curves) or update interval (at b.3) while all other parameters are held constant.

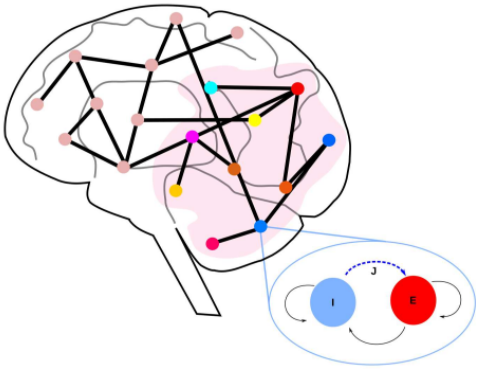


Figure 3.25: Use case 2 inspired by Deco *et al.* [2014] whole brain model: abstract representation of a brain model including 68 regions. A subset of the regions is selected (pink area). The zoom-in view depicts the model for each region consisting of two populations, one excitatory in red and one inhibitory in blue. Inhibitory connections to excitatory neurons in the same region (blue dashed arrow labeled *J*) are subject to structural plasticity.

shows how the network can traverse biologically inadmissible trajectories and still reach the target activity and that several simulations using different dynamics lead to the same target activity but different connectivity patterns. Using only target activity as a tuning parameter without visualization and steering can lead to arbitrarily selecting states. The network structure may be critically path-dependent, dependent upon parameters which are stochastic sequences or even dependent upon numerically unstable parameter functions.

Simple networks, as shown in this example, are frequently used but rarely with characterization of the parameter space. In absence of analytical methods to identify alternative solutions, steered visualization is a highly effective method for producing, observing, comparing and cataloging network configurations (**RQ2, RQ3**).

The second use case, inspired by the previous study of Deco *et al.* [2014], consists of a whole brain simulation using 68 interconnected brain regions each represented by a spiking network containing 200 neurons as shown in Figure 3.25. Each region contains two populations: one excitatory (80% of the total neurons in the region), and one inhibitory (20%). Each neuron initially receives 160 excitatory connections from the local excitatory population and only inhibitory connections are created during the simulation. The inter-regional connectivity (black lines between regions in Figure 3.25 is derived from structural data obtained by DTI. Each connection between regions is made between a single representative neuron in each excitatory population. Connections between regions are only excitatory. Additionally, all neurons receive independent background input from a Poisson generator producing spike trains with a rate of 11.9 kHz.

The CMV system is used with different inter-region global coupling factors G. By using the CMV system, it is evident that as G grows it is more difficult to bring all regions to a desired activity state and the standard deviation of the average firing rate increases. The CMV system also reveals which regions are crucial for stability as they have a higher interconnectivity to other ones. Figure 3.26 shows a comparison of the evolution of the firing rate and outgoing connections of four regions. Each peak shows an increment in the global coupling value G of 0.5 starting from a base value of 0.5. Regions 25 and 63 show large oscillations due to their high connectivity. Conversely, regions 0 and 10 rapidly reach a

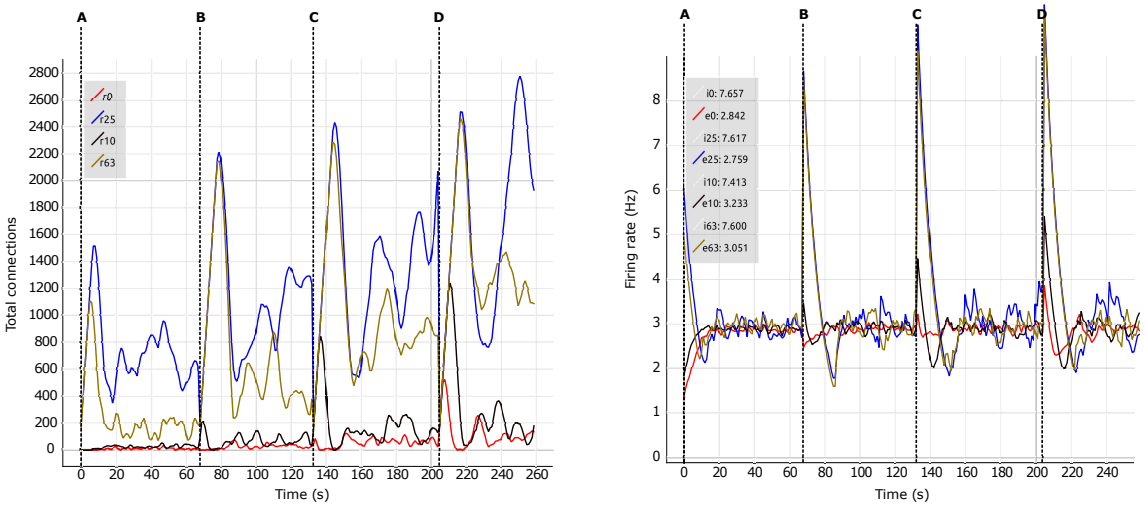


Figure 3.26: Number of connections (left) and firing rate (right) shown in comparison of four regions $(0, 10, 25, 63)$. Vertical dashed lines separate partition the simulation with differing values of $G =$ A) 0.5; B) 1.0; C) 1.5; D) 2.0. Regions are numbered from 0-67. Tags *eX* and *iX* identify excitatory and inhibitory populations while the number next to it denotes the current average firing rate.

stable state even for high values of G. The CMV system's capacity for detailed inspection allows to verify that all regions reach the desired average activity during simulation and drastically decreases turn-around times. In addition to the advantages in speed and use of computational resources, steering allows to gain more insight into the simulation.

In the first use case, the primary finding was that multiple connectivity configurations can result in the same activity profile. The second one identifies which regions are most critical for the overall network stability, as illustrated in Figure 3.26. In both cases, the CMV system supports researchers in sensitivity analysis, understanding the driving parameters of the model and assessing the relations between parameters and function.

To describe the typical workflow for both use cases, the first step is to determine which regions have one or more of the following characteristics (**R2-R5**):

1. the electrical activity is far away from the target activity and there is no correction tendency of the system, or the correction is too slow.
2. the electrical activity oscillates around the target activity and the oscillations are of equal or higher amplitude in each cycle.
3. the number of connections does not converge even though electrical activity is around the target activity.

Reaching the targeted stable state is indicated by all firing rate curves converging to the target activities while the connection curves are flattening to horizontal lines. This allows to effectively identify which regions are deviating from the target state and to correct the structural parameters according to the following criteria:

1. if the actual electrical activity is far away from the target activity, the growth rate ν for that region should be increased (**R6**).
2. if the actual electrical activity oscillates around the target activity, the growth rate ν

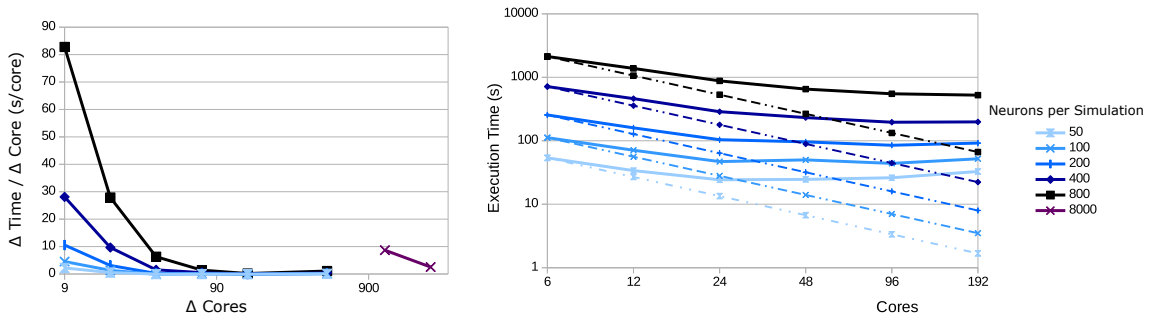


Figure 3.27: Execution time as a function of the number of compute nodes for varying numbers of neurons per population: 50, 100, 200, 400, 800, and 8,000. Dotted lines indicate ideal scaling while solid lines represent experimental results. The left plot shows the change in time per core added. Each point is the difference in execution time divided by the difference in cores for consecutive points in the plot next to it; points are placed at the midpoint between the measurements. The right plot depicts the execution time for each simulation with 8,000 neurons per population as total cores are varied.

for that region should be decreased in small increments and the value of η should be reduced to decrease the rate of change in the number of created and deleted synaptic elements around the target point ε (**R6-R7**).

3. if the number of connections is not converging, the growth rate ν of highly interconnected regions should be tuned down (**R7**). The update interval can be decreased to fasten a response of the control changes in the connectivity (**R8**). A shorter update interval allows smoother control but impacts the simulation's performance.

The resulting network state can be saved and used as a starting point for other parameter combinations, minimizing computations for similar values of G (**R9-R10**).

The CMV system scalability is dominated by the data gathering step at every update interval. Because the CMV system is independent of the network, the communication overhead can be neglected. However, to scale with increasing numbers of backend nodes, a more sophisticated data flow network is required. To simulate a larger number of populations with more neurons, the JURECA supercomputer at the Jülich Super Computing Centre was utilized. JURECA has 260 compute nodes with Intel Xeon E5 – 2680 v3 Haswell CPUs with 2×12 cores per CPU, 128 GB of RAM per node. To assess the speed-up obtained, measurements of the execution times for 50 updates of connectivity with an update interval of 100 ms were taken. Using a full node on JURECA, a speed-up of 2.94 was obtained compared to a workstation with 8 Intel Core i7 – 4710MQ CPUs @ 2.50 GHz and 16 GB of RAM. Figure 3.27 shows the strong scaling test. Scalability increases with the number of neurons per population as the number of synapses quadratically increases. This is because of the network size and spike distribution overhead: larger networks benefit more from increased compute-nodes [Jordan *et al.*, 2018]. This speed-up is achieved due to four factors:

1. it is not necessary to simulate the system for long times iteratively; instead, the modifications are performed on demand.

2. partial solutions can be reused for different global parameter combinations, resulting in the reduction of total computational costs.

3. the user can visualize the behavior of the system's observables with respect to individual parameters, allowing to isolate regions of interest and form a better understanding.
4. studying the transition points in the activity of the networks and interact with the tuning algorithms by visualizing their impact.

The connectivity solutions and paths to solutions are not unique, making knowledgeable exploration crucial. While the generation of connectivity based on empirical constraints, or experimental data leads to non-unique and physiologically implausible solutions, the ability to identify and explore subsets of the solution space is valuable to form an understanding of these systems. Using the CMV system, researchers can concentrate on only configurations of interest. The CMV system allows to characterize the distribution of representative models and enables a sensitivity analysis by visualizing the effect of connectivity changes on the dynamics of the neural system. More informative conclusions about the relationships between the control parameters and the resulting firing rate of each population can be drawn. In addition, the CMV system provides insight into how different types of synapses are created or modified to give rise to features in the dynamics. The CMV system reduces the turn-around times of exploring different connectivity configurations compared to simulating all possible parameter configurations and assess reasonable configurations in a later phase. Thus, the interactive analysis process can accomplish the following:

1. form an understanding of the implication of different parameter setups for each network model.
2. validate the network models.
3. define biologically meaningful populations for the simulation.
4. derive measures for the automatic or semi-automatic assessment of the models' behavior.

It is crucial to explore the distribution of paths to solutions instead of *a* possible solution satisfying a set of constraints. To form an understanding, interactive exploration of dynamic systems is an important tool to develop intuition and deriving mathematically robust descriptions. The presented CMV system is a way to move toward statistically validated conclusions as it allows to interactively assess the system, and can lead to automated sampling.

CHAPTER 4
DISCUSSION

In this thesis, the motivational factors for developing semantic-aware CMV systems were manifold. We started with visualizing neural activity data resulting from the simulation of neural models. These simulations use and produce a vast amount of heterogeneous data that need to be related to derive insight about the neural model's behavior. To this end, a useful visualization technique to relate these datasets is the utilization of CMVs. However, one of the major challenges of developing CMV systems is the potential rapid shift in the exploration process reflected in the scientist's workflow. This motivates a flexible software architecture adapting to the analysis needs.

The presented architecture solves this issue by defining services. Services encapsulate well-defined, fine-granular, reusable functional components for different use case scenarios by communicating via events. Events are send via inter-process communication or over a network. This characteristic allows to derive further benefits for the software architecture. Namely, service functionality can be used in different contexts or allow for the integration of already existing functionality outside the current software ecosystem, e.g., by aggregating statistical analysis capabilities as shown in the use case presented in Section 3.2. Also, it offers the means to scale the system across a variety computing resources and enables cross display system visualizations. Furthermore, the architecture enables *dynamic* coordination of views to be defined at runtime by users. In addition, it allows for a simple extensibility of additional views to a CMV system. This mechanism offers the flexibility to cover rapid changing workflows and adjust the CMV system to new user needs without modifying existing services. However, this flexibility requires the management of complex event-flows among the system's services which results from the communication via events. To compensate for this increased complexity, the architecture uses concepts from the semantic web: services' slots were augmented with semantic identifiers corresponding to individuals of semantic concepts defined in an ontology. The ontology is used by the *runtime system* to match compatible slots to reflect an event-flow network. This event-flow network resembles a workflow in the analysis process. In addition, event-flow networks can be recreated to effectively switch between workflows on demand. One of the main benefits of this approach is the mapping of coordination principles to slots enabling *dynamic* coordination among views at runtime. Dynamic coordination refers to the ability of establishing and revoking coordinations while services are running.

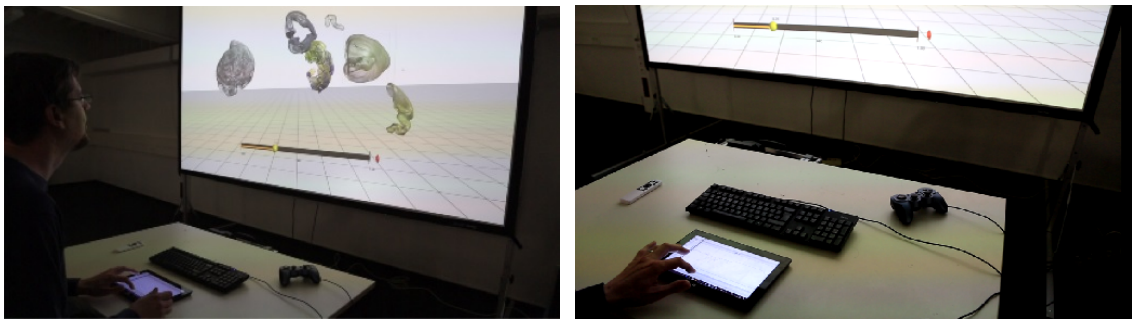


Figure 4.1: A user navigates in time and selects brain areas from a tablet running the *Time Navigator* and *Area Selection View*. The selection and time navigation is coordinated with the *Geometry View* running on a projection system in the background.

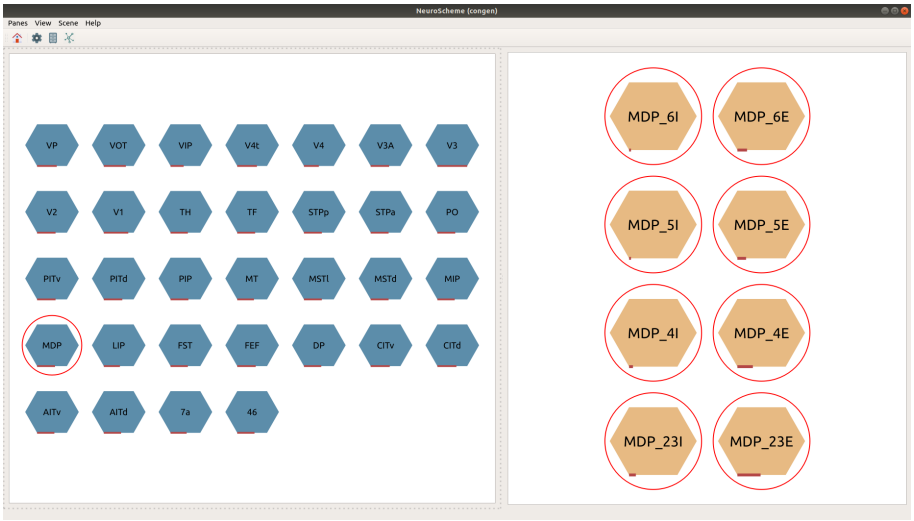


Figure 4.2: An overview of all brain areas of the simulation model is shown in the right pane. A progress bar at the bottom of each brain area's hexagon shows its number of active neurons. The current selection is hinted by a red circle encompassing the area in focus. The left pane is coordinated with the selection and shows the area's populations. A progress bar under each population shows the distribution of active neurons.

Using an EDA that allows communication via events over a network, enables services to run across multiple computing resources. In particular, this allows visualization services to utilize different display systems and permits hybrid 2D/3D cross platform visualization systems as seen in Figure 4.1. As previously discussed in Section 3.2, the *nett* framework enables the development of services across different programming languages and allows for a simple integration of existing tools to its ecosystem. To exemplify the applicability of the architecture's integration capabilities, *NeuroScheme*, introduced in Pastor *et al.* [2015] and Galindo *et al.* [2016], was integrated as an additional means to select brain areas and populations for the use case discussed in Section 3.1. Here, selections were coordinated with NeuroScheme to visualize and show a detailed overview of the simulated populations (see Figure 4.2). A key benefit of this semantic-aware CMV system is its extensibility. To integrate external tools, e.g., statistical analysis methods, existing services require no change. Merely new views are developed, started, and connected via slots according to the workflow needs. On top, visualizations can run on remote systems, utilizing more display-real-estate and computing resources.

The flexibility in adapting to different workflows resulting in the construction of a specialized CMV is realized by "wiring" services together to reflect the required functionality for the workflow. The functionality to wire services is also accessible in the commonly used modeling environment, i.e., Python. This simplifies assembling simulation output data for visualization and avoids further interruptions in the analysis workflow. In addition, it allows for scripting analysis tasks, e.g., jumping to and looping over a specific simulation time interval to closer inspect model behavior, study parameter influences in different simulations, or convey hypotheses about the data. For all that, this characteristic enables the development of fine-granular functionalities wrapped in services which fosters service re-usability in the construction of a PSE. However, it also increases the complexity of the event-flow network and its construction. On the other hand, developing services encapsulating a broader functionality spectrum, resulting in fewer slots and more independence from other services, restricts a service's re-usability. A decision on granularity should thus be guided by the following considerations:

1. how likely is the functionality exposed via a slot to be reused in different contexts?
2. is the functionality's computational complexity high enough that its distribution to more computational resources is warranted?
3. is the overhead introduced by wrapping functionality in an independent service small enough?

Since each service needs to be started, wired, and managed within the *runtime system*, exposing fine-granular functionality bears costs. Also, event communication equals data movement over a network, thus introducing additional latency to the system.

One limitation of the current architecture's implementation is its restricted scope to handle fault-tolerance. If a service crashes, the *runtime system* is unable to detect and react to this, however, restarted services automatically reconnect to the event-flow network. This limitation is overcome by introducing additional control structures within services via a supplementary control slot. For one, a mechanism to ping a running service can be used to detect if it is still operating. Secondly, services should announce that their startup process is complete and all slot operations are now ready to be wired and used. Thirdly, a mechanism to externally shutdown a service can be useful to cleanly switch to another PSE that does not need all currently running services. Fourthly, for services providing user interaction, a mechanism to detect which service currently holds user focus is useful to extend the system with collaboration features where multiple users can interact with the system concurrently. This would allow for the development of mitigation strategies for conflicting inputs to views, i.e., a locking mechanism that provides an exclusive interaction focus while neglecting input from users not holding a lock. This could lead to visualization systems that support a collaborative analysis process using the semantic-aware architecture. A further complexity is introduced by ontologies. While ontologies characterize, describe and formulate domains, their modeling is non-trivial. Multiple ways to express the same semantic concept exist and the principle to operate under an "open-world" assumption does not restrict the inference of facts known about the system. Service developers must be familiar with the modeled ontology, its core concepts, defined relations, entities, and their use within the system to extend the PSE. Restricting OWL's expressiveness by Domain-Specific Languages (DSLs) that in turn translate back into an ontology to allow for the use of formal logic to infer compatibility of slots in the system, might be a solution to provide a simpler, more direct access to the extensibility of

the system.

While the architecture's applicability was only shown for the presented use cases centering around neuroscientific data, it is not necessarily restricted in its scope to CMV systems in different visualization domains. The architecture's core principles can be applied in different visualization domains by developing and extending the system by custom-tailored views. To this end, the architecture's capabilities, namely, *dynamic coordination*, *cross display-system support*, *hybrid 2D/3D visualization interfaces*, *extensibility*, and *integration support* across programming language barriers, can be leveraged in different application domains.

CHAPTER 5

CONCLUSION AND OUTLOOK

In this thesis, I presented several visualization systems, using a common architecture, for the analysis of neural activity data. To develop these systems, I presented the overall motivation and resulting requirements. Common key requirements for such systems is extensibility to cope with heterogeneous data resulting from the simulation of neural systems and flexibility to cover rapidly shifting workflows. In addition, to discover relationships in the heterogeneous data, CMVs are used as the main visualization technique. To this end, an EDA was identified as a means to realize a PSE that enables extensibility by loosely coupled software services. I presented key characteristics of EDAs, namely, a common communication infrastructure and a mechanism to orchestrate services in order to reflect analysis workflows. This orchestration uses *semantic reasoning* to ease the construction of event-flow networks and is encapsulated in a service termed *runtime system*. Utilizing semantic reasoning allows for the definition of workflows, identifying compatible services for such, and establishing communication channels among services. Based on this, implementation of this architecture was carried out, resulting in a PSE. Novel contributions are the *dynamic coordination* capabilities and the possibility to add and remove views on demand at runtime. To show the architecture's applicability to cope with visualization requirements derived from use cases in neuroscience, several views and services were developed resulting in specific CMV systems constructed from the PSE: the first use case permitted the visual analysis of neural activity data resulting from a large-scale multi-area model of the macaque visual cortex. Here, I focused on the holistic coverage of heterogeneous data that was used to construct the model and its subsequently simulation by developing custom-tailored views. The second use case showcased the extensibility of the architecture by adding statistical analysis for spike trains. Statistical analysis methods were leveraged from an existing community tool developed by domain experts and demonstrated a practical approach to add interactive visualization methods, bridging the expertise of neuroscientists and visualization experts. The resulting views focused on the microscopic dataset resulting from the first use case, i.e., spike trains from the macaque visual cortex simulation and showed that the PSE is extensible to cover more complex analysis requirements. The last use case demonstrated how core functionality of EDAs can be leveraged to enable computational steering of neural network simulations. Here, I presented views to modify model parameters while a simulation is computed by utilizing the communication infrastructure developed in this thesis. In addition, views were

developed to visualize the model's dynamics.

The architecture's implementation covers all established requirements and provides the following benefits:

1. the flexibility to extend the PSE to new analysis workflows.
2. the ability to utilize hybrid display system, in particular the possibility to use immersive display systems in conjunction with traditional 2D UIs.
3. the semantically assisted dynamic coordination among views at runtime and the ability to revoke coordination.
4. multiple instantiation of services, enabling comparative analysis of datasets with synchronized user interaction.
5. the re-usability of services and views in different visualization tasks.

The presented architecture and its associated visualizations present a solution for CMV systems that assists in visual analysis tasks for modeling neural networks.

Future work should cover additional use cases, extending the palette of views and reusable services. Furthermore, the *runtime system* can be extending with convenience functionality to ease the construction of event-flow networks: a UI for sorting services by slots or filtering slots per semantic identifier is a useful addition. Using a DSL for the creation of CMV systems out of the PSE could ease the need for expert knowledge in formulating ontologies. To this end, a DSL could be used to specify a workflow along with needed services and its event-flow network. The DSL is then translated into an ontology and subsequently passed to the *runtime system* to construct the CMV system. Alternatively, an editor to visually create event-flow networks in the *runtime system* can be developed for a more user-friendly and intuitive way to construct a CMV system: a graph editor can be used to define a CMV system by selecting services which introduce vertices corresponding to the slots of the selected service. Edges can then be constructed between slots by selecting a vertex and highlighting or color-coding all compatible slots (represented by vertices) and draw a connection. This visual description could then be translated into an ontology and passed to the *runtime system* to realize the described event-flow network.

The inclusion of provenance tracking in the PSE is also interesting for future work. Here, services could be added recording all event-flow communication to track user interaction. Additionally, whenever the analysis workflow changes, the event-flow network is altered and the change in configuration stored in the ontology. A version control system can track these changes to create a provenance trail, recording how insight in the analysis task was established.

Developing methods to semantically describe interaction techniques utilized in views could be used to reason about compatible interaction metaphors across multiple display and input devices.

Finally, a powerful mechanism to extend the PSE is centered around modeling event transformation in the ontology. Here, semantic concepts can be introduced that describe how the event communication of one service via a slot can be transformed to a distinct concept for a different service slot. Then, the *runtime system* could use the specified transformation to automatically build a service realizing the desired transformation. When an

event-flow network is constructed that has no compatible slots for a service, all available transformations are checked to establish an event-flow to the required slot. This enables the extension of the PSE without directly implementing services and allows to relate the content of events to semantic concepts and relating concepts to each other, not only on a semantic level but on a functional one too.

APPENDIX – SCIENTIFIC CURRICULUM VITAE

Education

- **2022:** PhD Candidate in CS at the University Trier
- **2011–17:** PhD Candidate in CS at the RWTH Aachen University
- **2006–10:** M.Sc. in CS at the Bonn-Rhein-Sieg University of Applied Sciences
- **2003–06:** B.Sc. in CS at the Bonn-Rhein-Sieg University of Applied Sciences
- **2002:** Abitur at Joliot-Curie Gymnasium in Görlitz, Saxony

Research Focus

- Interactive Visualization Techniques
- Visualization of Neuroscientific Data
- Scientific Visualization in Virtual Environments & Realtime Computer Graphics
- Interaction Techniques in Virtual Environments
- Service Oriented Architectures
- Simulation Steering

Research Initiatives and Teaching

- Member Human Brain Project
- Member JARA, Jülich Aachen Research Alliance
- Member Supercomputing and Modeling for the Human Brain, Heimholtz Joint Lab
- Lecturer Data Analysis and Visualization, RWTH Aachen

- Master Thesis Supervision: Evaluating Interactive Diagrams for the Analysis of Neural Activity Data, Mohammadsobhan Moazemi Goodarzi, RWTH Aachen
- Bachelor Thesis Supervision: Darstellung von und Interaktion mit ergänzenden Informationen in Virtual Environments, Sven Horn, RWTH Aachen
- Seminar Supervision: Current Topics in Virtual Reality, RWTH Aachen

Awards and Invited Talks

- **2012: Honorable Mention,** IEEE Visualization 2012 for the Abstract "VisNEST - Interactive Analysis of Neural Activity Data"
- **2014: Donders Discussion** at the Donders Institute for Brain, Cognition and Behaviour, Nijmegen

Publications

- **Nowke, C.**, Diaz-Pier, S., Weyers, B., Hentschel, B., Morrison, A., Kuhlen, T. W., Peyser, A. (2018), Toward rigorous parameterization of underconstrained neural network models through interactive visualization and steering of connectivity generation, Frontiers in Neuroinformatics.
- Gebhardt, S., Petersen-Krau, T., Pick, S., Rausch, D., **Nowke, C.**, Knott, T., Schmitz, P., Zielasko, D., Hentschel, B., Kuhlen, T. W. (2016), Vista Widgets: A Framework for Designing 3D User Interfaces from Reusable Interaction Building Blocks, VRST 2016, 251-260.
- Weyers, B., **Nowke, C.**, Kuhlen T. W., Kousuke, M., Ogata, H. (2016), Web-based Interactive and Visual Data Analysis for Ubiquitous Learning Analytics, First International Workshop on Learning Analytics Across Physical and Digital Spaces, 65–69.
- **Nowke, C.**, Zielasko, D., Weyers, B., Peyser, A., Hentschel, B., Kuhlen, T. W. (2015), Integrating Visualizations into Modeling NEST Simulations, Frontiers in Neuroinformatics, Vol. 9.
- Weyers, B., **Nowke, C.**, Hänel, C. , Zielasko, D., Hentschel, B., Kuhlen, T. W. (2014) The Human Brain Project - Chances and Challenges for Cognitive Systems, Workshop Kognitive Systeme: Mensch, Teams, Systeme und Automaten.
- Van Albada S.J., Diesmann M., Eppler J. M., Hentschel B., Kuhlen T. W., **Nowke C.**, Reske M., Schmidt M. (2014), Modellierung und 3D-Visualisierung neuronaler Netzwerke in der Größenordnung des Gehirns, RWTH Themen 2:52-57.
- **Nowke, C.**, Hentschel, B., Kuhlen, T. W., Schmidt, M., van Albada, S. J., Eppler, J. M., Diesmann, M. (2013), Interactive visualization of brain-scale spiking activity, BMC Neuroscience, 14 (Suppl 1), P110.
- **Nowke, C.**, Schmidt, M., Albada, S. J. V., Eppler, J. M., Bakker, R., Diesmann, M., et al. (2013), VisNEST Interactive Analysis of Neural Activity Data, IEEE Symposium on Biological Data Visualization, 65-72.

BIBLIOGRAPHY

Abram, Greg, & Treinish, Lloyd. 1995. An extended data-flow architecture for data analysis and visualization. *Page 263 of: Proceedings of the 6th conference on Visualization'95*. IEEE Computer Society, Washington, DC, USA.

Arsiwalla, Xerxes D, Zucca, Riccardo, Betella, Alberto, Martinez, Enrique, Dalmazzo, David, Omedas, Pedro, Deco, Gustavo, & Verschure, Paul FMJ. 2015. Network dynamics with BrainX3: a large-scale simulation of the human brain network with real-time interaction. *Frontiers in Neuroinformatics*, **9**.

Assenmacher, Ingo, & Kuhlen, Torsten. 2008. The ViSTA Virtual Reality Toolkit. *Pages 23–28 of: Proceedings of the IEEE VR 2008 Workshop Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. Shaker Verlag.

Badam, Sriram Karthik, Fisher, Eli, & Elmqvist, Niklas. 2014. Munin: A Peer-to-Peer Middleware for Ubiquitous Analytics and Visualization Spaces. *IEEE Transactions on Visualization and Computer Graphics*, 1–1.

Bassett, Danielle S., & Gazzaniga, Michael S. 2011. Understanding complexity in the human brain. *Trends in Cognitive Sciences*, **15**(5), 200–209.

Becker, Richard A., & Cleveland, William S. 1987a. Brushing Scatterplots. *Technometrics*, **29**(2), 127–142.

Becker, Richard A, & Cleveland, William S. 1987b. Brushing scatterplots. *Technometrics*, **29**(2), 127–142.

Berger, Denise, Borgelt, Christian, Louis, Sebastien, Morrison, Abigail, & Grün, Sonja. 2010. Efficient identification of assembly neurons within massively parallel spike trains. *Computational Intelligence and Neuroscience*, **2010**, 1.

Binzegger, Tom, Douglas, Rodney J, & Martin, Kevan A C. 2004. A quantitative map of the circuit of cat primary visual cortex. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, **24**(39), 8441–53.

Borisyuk, Roman M., & Borisyuk, Galina N. 1997. Information coding on the basis of synchronization of neuronal activity. *Biosystems*, **40**(1-2), 3–10.

Bos, Hannah, Morrison, Abigail, Peyser, Alexander, Hahne, Jan, Helias, Moritz, Kunkel, Susanne, Ippen, Tammo, Eppler, Jochen Martin, Schmidt, Maximilian, Seeholzer, Alex, Djurfeldt, Mikael, Diaz, Sandra, Morén, Janne, Deepu, Rajalekshmi, Stocco, Teo, Deger, Moritz, Michler, Frank, & Plesser, Hans Ekkehard. 2015 (dec). *NEST 2.10.0*.

Boukhelifa, Nadia, & Rodgers, Peter J. 2003. A model and software system for coordinated and multiple views in exploratory visualization. *Information Visualization*, **2**(4), 258–269.

Bowman, Doug A, Kruijff, Ernst, LaViola Jr, Joseph J, & Poupyrev, Ivan. 2004. *3D user interfaces: theory and practice*. Addison-Wesley.

Buja, A., McDonald, J. A., Michalak, J., & Stuetzle, W. 1991 (Oct). Interactive data visualization using focusing and linking. *Pages 156–163, 419 of: Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on*.

Butz, Markus, & van Ooyen, Arjen. 2013. A simple rule for dendritic spine and axonal bouton formation can account for cortical reorganization after focal retinal lesions. *PLoS Comput Biol*, **9**(10), e1003259.

Card, Stuart K., Newell, Allen, & Moran, Thomas P. 1983. *The Psychology of Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

Carpenter, Loren. 1984. The A-Buffer, An Antialiased Hidden Surface Method. *Pages 103–108 of: Proceedings Siggraph*.

Childs, Hank, Brugger, Eric, Whitlock, Brad, Meredith, Jeremy, Ahern, Sean, Bonnell, Kathleen, Miller, Mark, Weber, Gunther H., Harrison, Cyrus, Fogal, Thomas, Garth, Christoph, S, Allen, Bethel, E. Wes, Durant, Marc, Camp, David, Favre, Jean M., Rübel, Oliver, Navrátil, Paul, Wheeler, Matthew, Selby, Paul, & Vivodtzev, Fabien. 2011. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. *In: In Proceedings of SciDAC*.

Cleveland, William S, *et al.* 1985. *The elements of graphing data*. Wadsworth Advanced Books and Software Monterey, CA.

Connors, Barry W, & Gutnick, Michael J. 1990. Intrinsic firing patterns of diverse neocortical neurons. *Trends in Neurosciences*, **13**(3), 99–104.

Cubitt, Toby S, Eisert, Jens, & Wolf, Michael M. 2012. Extracting dynamical equations from experimental data is NP hard. *Physical review letters*, **108**(12), 120503.

Davison, Andrew. 2012. Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering*, **14**(4), 48–56.

Davison, Andrew P, Brüderle, Daniel, Eppler, Jochen M, Kremkow, Jens, Muller, Eilif, Pecevski, Dejan, Perrinet, Laurent, & Yger, Pierre. 2009. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, **2**(11).

Decker, Stefan, Melnik, Sergey, Van Harmelen, Frank, Fensel, Dieter, Klein, Michel, Broekstra, Jeen, Erdmann, Michael, & Horrocks, Ian. 2000. The semantic web: The roles of XML and RDF. *IEEE Internet computing*, **4**(5), 63–73.

Deco, Gustavo, Ponce-Alvarez, Adrián, Mantini, Dante, Romani, Gian Luca, Hagmann, Patric, & Corbetta, Maurizio. 2013. Resting-state functional connectivity emerges from structurally and dynamically shaped slow linear fluctuations. *The Journal of Neuroscience*, **33**(27), 11239–11252.

Deco, Gustavo, Ponce-Alvarez, Adrián, Hagmann, Patric, Romani, Gian Luca, Mantini, Dante, & Corbetta, Maurizio. 2014. How local excitation–inhibition ratio impacts the whole brain dynamics. *The Journal of Neuroscience*, **34**(23), 7886–7898.

Diaz Pier, Sandra, Naveau, Mikael, Butz-Ostendorf, Markus, & Morrison, Abigail. 2016. Automatic generation of connectivity for large-scale neuronal network models through structural plasticity. *Frontiers in Neuroanatomy*, **10**(57).

Eichelbaum, Sebastian, Hlawitschka, Mario, Wiebel, Alexander, & Scheuermann, Gerik. 2010. OpenWalnut–An open-source visualization system. *Pages 67–78 of: Proceedings of the 6th High-End Visualization Workshop*.

Ellis, G., Dix, A., & Bertini, E. 2005. The Sampling Lens: making sense of saturated visualisation. *Proceedings of CHI'2005, ACM Press*, 1351–1354.

Eppler, Jochen M, Helias, Moritz, Muller, Eilif, Diesmann, Markus, & Gewaltig, Marc-Oliver. 2009. PyNEST: a convenient interface to the NEST simulator. *Frontiers in Neuroinformatics*, **2**(12).

Fekete, J.-D., & Plaisant, C. 2002. Interactive information visualization of a million items. *IEEE Symposium on Information Visualization*.

Felleman, D. J., & Van Essen, D. C. 1991. Distributed Hierarchical Processing in the Primate Cerebral Cortex. *Cerebral Cortex*, **1**(1), 1–47.

Folk, M., Cheng, A., & Yates, K. 1999 (Nov.). HDF5: A file format and I/O library for high performance computing applications. *In: Proceedings of SC'99*.

Frost, C. 2010. Challenges and opportunities for autonomous systems in space. *In: Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2010 Symposium*.

Galindo, Sergio E., Toharia, Pablo, Robles, Oscar D., & Pastor, Luis. 2016. ViSimpl: Multi-View Visual Analysis of Brain Simulation Data. *Frontiers in Neuroinformatics*, **10**, 44.

Gebhardt, Sascha, Pick, Sebastian, Leithold, Franziska, Hentschel, Bernd, & Kuhlen, Torsten. 2013. Extended Pie Menus for Immersive Virtual Environments. *IEEE Transaction on Visualization and Computer Graphics*, **19**(4), 644–51.

Gewaltig, Marc-Oliver, & Diesmann, Markus. 2007. NEST (NEural Simulation Tool). *Scholarpedia*, **2**(4), 1430.

Gray, Charles M, & McCormick, David A. 1996. Chattering cells: superficial pyramidal neurons contributing to the generation of synchronous oscillations in the visual cortex. *Science*, **274**(5284), 109–113.

Grün, Sonja, & Rotter, Stefan. 2010. *Analysis of parallel spike trains*. Springer.

Helias, Moritz, Kunkel, Susanne, Masumoto, Gen, Igarashi, Jun, Eppler, Jochen Martin, Ishii, Shin, Fukai, Tomoki, Morrison, Abigail, & Diesmann, Markus. 2012. Supercomputers Ready for Use as Discovery Machines for Neuroscience. *Frontiers in Neuroinformatics*, **6**(26).

Henderson, Amy. 2004. *The ParaView Guide: A Parallel Visualization Application*. Kitware.

Hentschel, Bernd, Tedjo, Irene, Probst, Markus, Wolter, Marc, Behr, Marek, Bischof, Christian, & Kuhlen, Torsten. 2008. Interactive Blood Damage Analysis for Ventricular Assist Devices. *Transactions on Visulaization and Computer Graphics*, **14**(6), 1515–1522.

Hernando, J.B., Schürmann, F., & Pastor, L. 2012. Towards Real-Time Visualization of Detailed Neural Tissue Models: View Frustum Culling for Parallel Rendering. *Pages 25–32 of: Proceedings of the IEEE Symposium on Biological Data Visualization (BioVis)*.

Hines, Michael L, & Carnevale, Nicholas T. 1997. The NEURON simulation environment. *Neural computation*, **9**(6), 1179–1209.

Hodgkin, Alan L, & Huxley, Andrew F. 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, **117**(4), 500–544.

Hummel, Mathias, Garth, Christoph, Hamann, Bernd, Hagen, Hans, & Joy, Kenneth I. 2010. IRIS: Illustrative Rendering for Integral Surfaces. *Transactions on Visulaization and Computer Graphics*, **16**(6), 1319–1328.

Issarny, Valérie, Saridakis, Titos, & Zarras, Apostolos. 1998. Multi-view Description of Software Architectures. *Pages 81–84 of: Proceedings of the Third International Workshop on Software Architecture*. ISAW '98. New York, NY, USA: ACM.

Izhikevich, Eugene M. 2004. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, **15**(5), 1063–1070.

Izhikevich, Eugene M, *et al.* 2003. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, **14**(6), 1569–1572.

Jordan, Jakob, Ippen, Tammo, Helias, Moritz, Kitayama, Itaru, Sato, Mitsuhisa, Igarashi, Jun, Diesmann, Markus, & Kunkel, Susanne. 2018. Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers. *Frontiers in Neuroinformatics*, **12**, 2.

Kalawsky, Roy S. 2009. Gaining greater insight through interactive visualization: A human factors perspective. *Pages 119–154 of: Trends in Interactive Visualization*. Springer.

Kamil Rajdl, Petr Lansky. 2014. Fano factor estimation. *Mathematical Biosciences and Engineering*, **11**(1551-0018-2014-1105), 105.

Kasiński, Andrzej, Pawlowski, Juliusz, & Ponulak, Filip. 2009. SNN3DViewer - 3D Visualization Tool for Spiking Neural Network Analysis. *Pages 469–476 of: Proceedings of the International Conference on Computer Vision and Graphics: Revised Papers*. ICCVG 2008.

Keefe, Daniel F., Ewert, Marcus, Ribarsky, William, & Chang, Remco. 2009. Interactive Coordinated Multiple-View Visualization of Biomechanical Motion Data. *IEEE Transactions on Visualization and Computer Graphics*, **15**(6), 1383–1390.

Keim, Daniel, Andrienko, Gennady, Fekete, Jean-Daniel, Görg, Carsten, Kohlhammer, Jörn, & Melançon, Guy. 2008. *Visual Analytics: Definition, Process, and Challenges*. Berlin, Heidelberg: Springer Berlin Heidelberg. Pages 154–175.

Krasner, Glenn E., & Pope, Stephen T. 1988. A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80. *J. Object Oriented Program.*, **1**(3), 26–49.

Kreuz, Thomas, Haas, Julie S., Morelli, Alice, Abarbanel, Henry D I, & Politi, Antonio. 2007. Measuring spike train synchrony. *Journal of Neuroscience Methods*, **165**, 151–161.

Kreuz, Thomas, Chicharro, Daniel, Houghton, Conor, Andrzejak, Ralph G, & Mormann, Florian. 2013. Monitoring spike train synchrony. *Journal of Neurophysiology*, **109**(5), 1457–72.

Kreuz, Thomas, Mulansky, Mario, & Bozanic, Nebojsa. 2015. SPIKY: A graphical user interface for monitoring spike train synchrony. *Journal of Neurophysiology (Submitted)*.

Laha, B., Sensharma, K., Schiffbauer, J. D., & Bowman, D. A. 2012. Effects of Immersion on Visual Analysis of Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, **18**(4), 597–606.

Lamme, Victor AF, Super, Hans, Spekreijse, Henk, *et al.* 1998. Feedforward, Horizontal, and Feedback Processing in the Visual Cortex. *Current opinion in neurobiology*, **8**(4), 529–535.

LeCun, Yann, Bengio, Yoshua, & Hinton, Geoffrey. 2015. Deep learning. *Nature*, **521**(7553), 436–444.

Lewis, Clayton, Rieman, John, & Blustein, Amended J. 1993. *Task-Centered User Interface Design: A practical introduction.*

Markov, N. T., Ercsey-Ravasz, M. M., Ribeiro Gomes, A. R., Lamy, C., Magrou, L., Vezoli, J., Misery, P., Falchier, A., Quilodran, R., Gariel, M. A., Sallet, J., Gamanut, R., Huissoud, C., Clavagnier, S., Giroud, P., Sappey-Marinier, D., Barone, P., Dehay, C., Toroczkai, Z., Knoblauch, K., Van Essen, D. C., & Kennedy, H. 2012. A Weighted and Directed Interareal Connectivity Matrix for Macaque Cerebral Cortex. *Cerebral Cortex*.

Markov, NT, Misery, P, Falchier, A, Lamy, C, Vezoli, J, Quilodran, R, Gariel, MA, Giroud, P, Ercsey-Ravasz, M, Pilaz, LJ, *et al.* 2011. Weight consistency specifies regularities of macaque cortical networks. *Cerebral Cortex*, **21**(6), 1254–1272.

Matkovic, Kresimir, Freiler, Wolfgang, Gracanin, Denis, & Hauser, Helwig. 2008a (jul). ComVis: a Coordinated Multiple Views System for Prototyping New Visualization Technology. *Pages – of: Proceedings of the 12th International Conference Information Visualisation*.

Matkovic, Kresimir, Gracanin, Denis, Jelovic, Mario, & Hauser, Helwig. 2008b. Interactive visual steering-rapid visual prototyping of a common rail injection system. *IEEE Transactions on Visualization and Computer Graphics*, **14**(6), 1699–1706.

Matković, Kreŝimir, Gračanin, Denis, Splechtna, Rainer, Jelović, Mario, Stehno, Benedikt, Hauser, Helwig, & Purgathofer, Werner. 2014. Visual analytics for complex engineering systems: Hybrid visual steering of simulation ensembles. *IEEE transactions on visualization and computer graphics*, **20**(12), 1803–1812.

McDonald, John Alan, Stuetzle, Werner, & Buja, Andreas. 1990. Painting multiple views of complex objects. *Pages 245–257 of: ACM SIGPLAN Notices*, vol. 25. ACM.

McGovern, James, Sims, Oliver, Jain, Ashish, & Little, Mark. 2006. Event-driven architecture. *Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations*, 317–355.

Michelson, Brenda M. 2006. Event-driven architecture overview. *Patricia Seybold Group*, **2**.

Neumann, John von. 1958. *The Computer and the Brain*. USA: Yale University Press.

Newman, Sam. 2015. *Building Microservices: Designing Fine-Grained Systems*. 1st edn. O'Reilly Media.

North, Chris, & Shneiderman, Ben. 1997. *A Taxonomy of Multiple Window Coordination*.

North, Chris, & Shneiderman, Ben. 2000. Snap-together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata. *Pages 128–135 of: Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '00. New York, NY, USA: ACM.

Nowke, Christian, Schmidt, Maximilian, Albada, Sacha J Van, Eppler, Jochen M., Bakker, Rembrandt, Diesmann, Markus, Hentschel, Bernd, & Kuhlen, Torsten. 2013. VisNEST – Interactive Analysis of Neural Activity Data. *IEEE Symposium on Biological Data Visualization (BioVis)*, 65–72.

Nowke, Christian, Zielasko, Daniel, Weyers, Benjamin, Hentschel, Bernd, Peyser, Alexander, & Kuhlen, Torsten. 2015. Integrating Visualizations into Modeling NEST Simulations. *Frontiers in Neuroinformatics*, **9**(29).

Nowke, Christian, Diaz-Pier, Sandra, Weyers, Benjamin, Hentschel, Bernd, Morrison, Abigail, Kuhlen, Torsten W., & Peyser, Alexander. 2018. Toward Rigorous Parameterization of Underconstrained Neural Network Models Through Interactive Visualization and Steering of Connectivity Generation. *Frontiers in Neuroinformatics*, **12**, 32.

Parker, Steven G, Miller, Michelle, Hansen, Charles D, & Johnson, Christopher R. 1998. An integrated problem solving environment: The SCIRun computational steering system. *Pages 147–156 of: PROCEEDINGS OF THE HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES*, vol. 31. Citeseer.

Pastor, Luis, Mata, Susana, Toharia, Pablo, Bayona, Sofia, Brito, Juan P., & Garcia-Cantero, Juan J. 2015. NeuroScheme: Efficient multiscale representation for the visual exploration of morphological data in the human brain neocortex. *Pages 117–125 of: Proc. of Congreso Español de Informática Gráfica*. Eurographics Association.

Pick, Sebastian, Hentschel, Bernd, Wolter, Marc, Tedjo-Palczynski, Irene, & Kuhlen, Torsten. 2010. Automated Positioning of Annotations in Immersive Virtual Environments. *Pages 1–8 of: Proceedings of the Joint Virtual Reality Conference of EuroVR - EGVE - VEC*.

Plesser, Hans E, Eppler, Jochen M, Morrison, Abigail, Diesmann, Markus, & Gewaltig, Marc-Oliver. 2007. Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. *Pages 672–681 of: Euro-Par 2007 parallel processing*. Springer.

Potjans, Tobias C., & Diesmann, Markus. 2012. The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model. *Cerebral Cortex*. Online first.

Quiroga, R Quian, Kreuz, T, & Grassberger, Peter. 2002. Event synchronization: a simple and fast method to measure synchronicity and time delay patterns. *Physical Review E*, **66**(4), 041904.

Raja, Dheva, Bowman, Doug A, Lucas, John, & North, Chris. 2004. Exploring the Benefits of Immersion in Abstract Information Visualization. *Pages 61–69 of: Proceedings of the 8th Immersive Projection Technology Workshop*.

Roberts, Jonathan C. 2007. State of the Art: Coordinated & Multiple Views in Exploratory Visualization. *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization*, July, 61–71.

Ryu, Y. S., Yost, B., Convertino, G., Chen, J., & North, C. 2003. Exploring Cognitive Strategies for Integrating Multiple-View Visualizations. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, **47**(3), 591–595.

Schmidt, Maximilian, Bakker, Rembrandt, Diesmann, Markus, & van Albada, Sacha. 2014. A spiking multi-area network model of macaque visual cortex. *In: Annual meeting of the SfN*. Computational and Systems Neuroscience, Osaka, Japan.

Schmitt, Oliver, & Eipert, Peter. 2012. neuroVIISAS: Approaching Multiscale Simulation of the Rat Connectome. *Neuroinformatics*, **10**, 243–267.

Schuecker, J., Schmidt, M., van Albada, S., Diesmann, M., & Helias, M. 2015. Fundamental activity constraints lead to specific interpretations of the connectome. *ArXiv e-prints 1509.03162*, Sept.

Sedlmair, Michael, Heinzl, Christoph, Bruckner, Stefan, Piringer, Harald, & Möller, Torsten. 2014. Visual parameter space analysis: A conceptual framework. *IEEE transactions on visualization and computer graphics*, **20**(12), 2161–2170.

Segev, Ronen, Baruchi, Itay, Hulata, Eyal, & Ben-Jacob, Eshel. 2004. Hidden Neuronal Correlations in Cultured Networks. *Phys. Rev. Lett.*, **92**(Mar), 118102.

Sejnowski, Terrence Joseph, Koch, Christof, & Churchland, Patricia Smith. 1988. Computational neuroscience. *Science*, **241**(4871), 1299–1306.

Shinomoto, Shigeru, Kim, Hideaki, Shimokawa, Takeaki, Matsuno, Nanae, Funahashi, Shintaro, Shima, Keisetsu, Fujita, Ichiro, Tamura, Hiroshi, Doi, Taijiro, Kawano, Kenji, Inaba, Naoko, Fukushima, Kikuro, Kurkin, Sergei, Kurata, Kiyoshi, Taira, Masato, Tsutsui, Ken-Ichiro, Komatsu, Hidehiko, Ogawa, Tadashi, Koida, Kowa, Tanji, Juna, & Toyama, Keisuke. 2009. Relating Neuronal Firing Patterns to Functional Differentiation of Cerebral Cortex. *PLoS Computational Biology*, **5**(7), 1.

Shneiderman, B. 1996. The eyes have it: a task by data type taxonomy for information visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*, 336–343.

Shneiderman, Ben. 1994. Dynamic queries for visual information seeking. *IEEE software*, **11**(6), 70–77.

Sousa, Mafalda, & Aguiar, Paulo. 2014. Building, simulating and visualizing large spiking neural networks with NeuralSyns. *Neurocomputing*, **123**(Jan.), 372–380.

Spence, Robert. 2001. *Information visualization*. Vol. 1. Springer.

Stein, RB. 1967. The frequency of nerve action potentials generated by applied currents. *Proceedings of the Royal Society of London B: Biological Sciences*, **167**(1006), 64–86.

Stephan, KE, Kamper, L., Bozkurt, A., Burns, G. A., Young, M. P., & Kotter, R. 2001a. Advanced database methodology for the Collation of Connectivity data on the Macaque brain (CoCoMac). *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, **356**(1412), 1159–86. 0962-8436 Journal Article.

Stephan, K.E., Kamper, L., Bozkurt, A., Burns, G.A.P.C., Young, M.P., & Kötter, R. 2001b. Advanced Database Methodology for the Collation of Connectivity Data on the Macaque Brain (CoCoMac). *Philosophical Transactions of the Royal Society London, Series B*, **356**, 1159–1186.

Tufte, Edward R. 1991. Envisioning information. *Optometry & Vision Science*, **68**(4), 322–324.

Velleman, Paul F, & Velleman, Agelia Y. 1988. The Datadesk Handbook. *Odesta Corporation*, **4048**.

von Kapri, Anette, Potjans, Tobias C., Kuhlen, Torsten, & Diesmann, Markus. 2011. A Virtual Reality Exploration Tool for Multi-Scale Data from Brain-Scale Simulations. *Frontiers in Neuroinformatics*.

Wang, Jia, & Lindeman, Robert. 2014. Coordinated 3D Interaction in Tablet- and HMD-based Hybrid Virtual Environments. *Pages 70–79 of: Proceedings of the 2Nd ACM Symposium on Spatial User Interaction*. SUI '14. New York, NY, USA: ACM.

Wang, Jia, & Lindeman, Robert. 2015. Coordinated hybrid virtual environments: Seamless interaction contexts for effective virtual reality. *Computers & Graphics*, **48**, 71 – 83.

Wang, Xiao Hang, Zhang, Da Qing, Gu, Tao, & Pung, Hung Keng. 2004. Ontology based context modeling and reasoning using OWL. *Pages 18–22 of: Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE Computer Society, Washington, DC, USA.

Wang Baldonado, Michelle Q., Woodruff, Allison, & Kuchinsky, Allan. 2000. Guidelines for Using Multiple Views in Information Visualization. *Pages 110–119 of: Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '00. New York, NY, USA: ACM.

Ware, C., & Franck, G. 1994. Viewing a graph in a virtual reality display is three times as good as a 2D diagram. *Pages 182–183 of: Proceedings of 1994 IEEE Symposium on Visual Languages*. IEEE Comput. Soc. Press.

Weaver, Chris. 2004. Building highly-coordinated visualizations in Improvise. *Pages 159–166 of: IEEE Symposium on Information Visualization, 2004*. IEEE Computer Society, Austin, TX, USA.

Yi, Ji Soo, ah Kang, Youn, Stasko, John T, & Jacko, Julie A. 2007. Toward a deeper understanding of the role of interaction in information visualization. *Visualization and Computer Graphics, IEEE Transactions on*, **13**(6), 1224–1231.

Zaytsev, Yury V., & Morrison, Abigail. 2014. CyNEST: a maintainable Cython-based interface for the NEST simulator. *Frontiers in Neuroinformatics*, **8**(23).