Master Thesis

TVB-EduPack

An interactive learning and scripting platform for *The Virtual Brain*

Author:
Henrik Matzke
hmatzke@inf.fu-berlin.de

Supervisor:
Prof. Dr. Raúl Rojas
Dr. Tim Landgraf
Advisor:
Dr. med. Petra Ritter

September 20, 2014

Freie Universität Berlin
Department of Mathematics and Computer Science
Intelligent Systems and Robotics Lab

Declaration of Academic Integrity

I hereby confirm that the present thesis

TVB-EduPack
An interactive learning and scripting platform for *The Virtual Brain*

is solely my own work and that if any text passages or diagrams from books, papers, the Web or other sources have been copied or in any other way used, all references – including those found in electronic media – have been acknowledged and fully cited.

Berlin, September 20, 2014

Henrik Matzke

Abstract

With the growing interest to understand processes and function of the human brain, several interesting research projects have started all over the world. They try to take advantage of the increasing computing power to open new possibilities in neuroscience. Among them is The Virtual Brain (TVB), a neuroinformatics platform that simulates the dynamics of large-scale neural populations through computational models.

Despite a set of features that make TVB understandable for the user, it is still a complex simulation tool. Specially, when considering users without deep knowledge in neuroscience and its computational principles. Therefore, in order to facilitate TVB's usability of the Graphical User Interface (GUI), an educational tool that helps the user to learn interactively, was required. In this thesis, I will present an add-on for The Virtual Brain called TVB-EduPack. It contains user-driven tutorials and adds a new feature that allows the creation of new tutorials by providing an easy-to-use scripting interface. Additionally, through a predefined and extendable set of user actions, it is possible for instance to use and display pre-computed simulation results.

This application is expected to improve significantly the manageability to the software of TVB. Specially, it can attract the attention of both, novice users interested in neurosciences as well as advanced researchers, who could make use of the tutorial authoring tool in order to publish and share their results with TVB's community.

Contents

List of Figures

1 Introduction

As humans, we can identify galaxies light years away, we can study particles smaller than an atom. But we still haven't unlocked the mystery of the three pounds of matter that sits between our ears. — U.S. President Barack Obama, 2013

The Human Brain

Billions of neurons form and fire in the most complex system we know – the human brain. There are approximately 86 billion nerve cells (ca. 100 * 10^9) and each of them has around 10.000 (10^4) connections to other nerve cells (summing up to 100 * 10^{13}). Considering that each of these cells can have two states – firing and non-firing, there are at least about $2^{(10^{15})}$ different states, as many as stars in the universe.

Computer Brain Models

There are different approaches that have all the same goal: understanding the human brain in more detail – not only chemical, but also functional, as well as cognition and memory [[4], [7]]. Brodmann published and analyzed the brain based on its cytoarchitecture[1] in 1909 [2]. He distinguished 43 different areas (parcels) on the cerebral cortex by characteristic organization of cells under the microscope. 100 years later, researchers are still working on projects that aim at mapping the human brain [1] based on cytoarchitecture, but also supported by neuroimaging techniques to gather information about structural and functional connectivity, as well as brain dynamics. The difficulty of creating a human brain map, or human brain atlas, is that every brain is individual and inhomogeneous; regions differ in structure and size and can even change over time. The Institute of NeuroScience and Medicine (INM-1) Jülich is developing probability maps of the human brain[2], with respect to the inter-individual variability they create a dynamic architectonic brain mapping system with probabilities for each voxel. Each voxel of size 1mm^3 still contains millions of neurons and connecting synapses [5].

[1] Cytoarchitecture describes e.g. the process of distinguishing the brain in different areas through differences in the cellular composition that can be detected under the microscope.

[2] Project Website: http://www.fz-juelich.de/inm/inm-1/DE/Forschung/_docs/BrainMapping

Macroscopic and Mesoscopic Approaches

One way to analyze and understand the function of the brain could be to model and simulate each neuron and all connecting synapses of the brain. That is exactly the goal of the Human Brain project[3], which has been funded with 1.19 billion € from the E.U. for ten years. The Human Brain project aims at simulating all neurons and their connecting synapses on analog non-clocked neuromorphic supercomputer chips [13]. Beside the enormous computational complexity, which is not possible with clocked computers[4], an additional interesting part is missing: how do neurons or neuronal populations generate complex behavior [15]? To reduce the computational complexity and details, one can take the brain connectivity from microscale, where the anatomical structures and physiological processes are analyzed, to a more abstract level – the mesoscopic and macroscale. Within the macroscale connectivity, the brain is abstracted into interconnected distinct brain regions that form the variable anatomical structure [6]. The aforementioned technology of macroscopic connectivity is an integral feature of The Virtual Brain.

1.1 The Virtual Brain Project

Introduction

The Virtual Brain is a neuroinformatics platform available for everyone. For people with little computer background, who require a software that contains everything in one package, a TVB stand-alone distribution is provided. This distribution is available for all major operating systems, including Windows, Mac and Linux, and consists of all packages used by TVB, such as Python, CherryPy, Genshi, NumPy, SciPy and more, so that the simulator and TVB framework may operate in any web browser. The 2D and 3D visualizations are implemented with WebGL, and the data and project management is done with SQL. This high-end solution is the easiest way for users like clinicians, as well as neuroscientists, who want to use TVB and work with a graphical user interface.

Another way to use TVB from scratch is to download the sources from a github repository, and install all required packages manually or with a package manager. The Virtual Brain is licensed under GPL v.2, a general public license which allows anyone to use, copy, distribute and modify the software under certain conditions. Installation tutorials, User Guides, and iPython notebooks can be found on the provided documentation page of TVB[5].

TVB is distributed with a console- and a graphical-web-based interface. The main focus in this paper lies on The Virtual Brain distribution with graphical user interface.

[3]https://www.humanbrainproject.eu/

[4]Neuromorphic chips are an alternative computing method that contains electronic analog circuits to mimic neuro-biological architectures.

[5]The Virtual Brain documentation: http://docs.thevirtualbrain.org/index.html

TVB Architecture

One of the big advantages of The Virtual Brain is that it combines data and project management with data simulation, visualizations, analysis and further data processing possibilities. More details about the software architecture of TVB can be found TVB User Guide [26].

The Virtual Brain provides a graphical and a console user interface. Both systems are connected and communicate with TVB core modules. In the graphical user interface, a web server interconnects with the core modules and the web GUI. Task of TVB core is to manage the data flow with its datatypes and adapters that are read and written from the storage backend, and exchanged with several integrated libraries. The abstraction of datatypes in TVB core facilitates the data interchange quite efficiently [28].

1.1.1 Basic Functionalities of The Virtual Brain

Large-Scale Network Model

The Virtual Brain is a full brain simulation tool. It works with the concept of brain parcellations divided into cortical and subcortical regions that are conceptualized as connected mesoscopic nodes in a large-scale network model. The dynamics of each node captures typical features of neuronal population activity. This is implemented upon other terms with neurocomputational models inside the nodes of the large-scale network. It would be even possible to run the simulations on the nodes of the 3d mesh of a cortical surface.

Figure 1.1: Abstraction from Human Brain to the parcellated Brain, with cortical and subcortical Regions, conceptualized to connected mesoscopic Nodes in a Large-Scale Network.

Other dynamic factors could be signal transmission delays or noise [11]. This is important for the excitatory or inhibitory influence of single nodes onto the whole complex network. Distances between nodes of the network and capacities, which are derived from diffusion-weighted magnetic resonance imaging (*dw-MRI*), are saved within a connectivity dataset. The process of dw-MRI, also referred to as diffusion tensor imaging (*DTI*), is the diffusion process of molecules in the brain, which can be recorded noninvasively.

Tractography techniques take advantage of the high-resolution data, and can compute non-directed fibers out of the anisotropic diffusion of the molecules. Beside the consistent diffusion in all three dimensions, the anisotropic diffusion is probably due to little boundaries like cell membranes or myelin sheaths of axons, which transmit messages to other connected neurons.

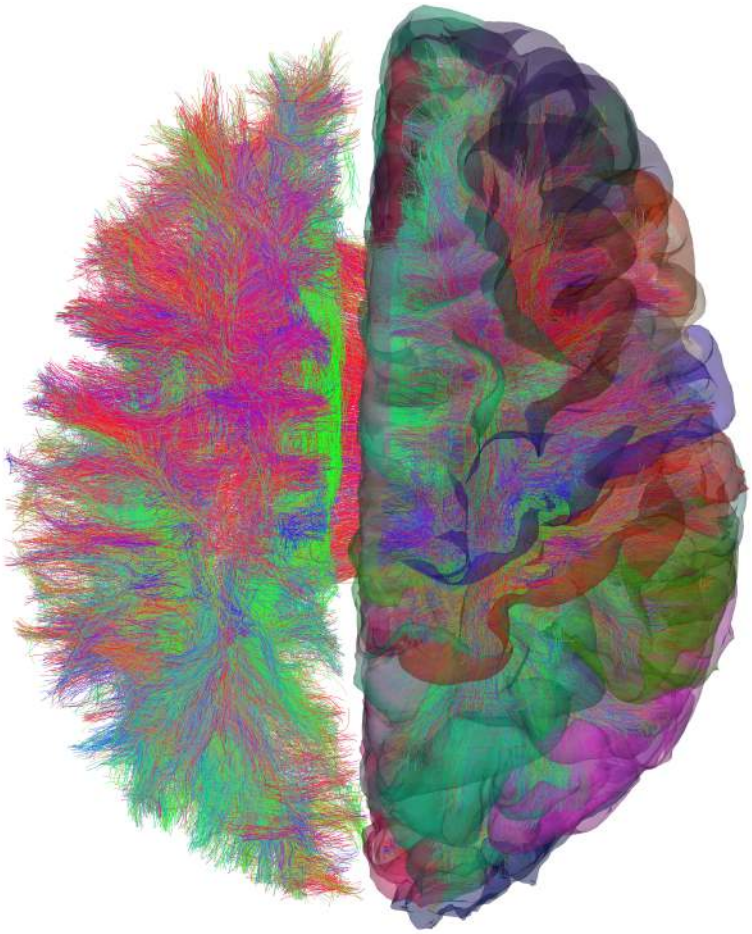

Figure 1.2: The Figure shows the Tractography Data of a Human Brain with a visible Surface on the right Hemisphere. The Data was retrieved during studies at the BrainModes Group, PD Dr. Ritter [24].

Connectivity Dataset

The generated and provided structural connectivity graph is defined through node positions, connection weights and signal transmission delays. Usually, a dataset is based on real anatomical data that enables the user to simulate subject specific full brain neuronal activity. The usage of individual large scale connectivity data, based on T1 anatomical

MRI scans and a hemodynamical model, allows the prediction of individualized EEG[6], MEG[7] and fMRI[8] under some physiological or pathological conditions [[8], [27]]. TVB offers more than ten implemented local dynamic models that all have different characteristics and simulate typical mesoscopic population dynamics within each node of the large-scale network. Additionally every model can be adapted with its own configuration parameters and all local dynamic model parameter settings can be set either for the whole network or on single nodes of the large-scale network [23].

Large-Scale Brain Network Equation in TVB

The large-scale network equation in TVB describes a stochastic differential model of a graph of connected neural populations, see Figure 1.3. With the distinction of short-range and long-range inputs, the equation distinguishes between local and global connectivity. Within the local connectivity, which is in the range of centimeters, the signal transmission is instantaneous and does not have a delay. The long-range input is in the range of 10 centimeters and has a signal transmission delay, which is dependent on the distance *(x-x')* and transmission speed v.

The equation also captures stimuli and noise in The Virtual Brain. A stimuli is defined over its spatial position x and a course of time t, while the noise can be configured independently for its spatial and temporal correlations [12].

1.1.2 TVB User Interfaces

The Virtual Brain comes with a console interface for experienced users, who have background experience in neuroscientific computing. But on the other hand, The Virtual Brain is a framework for the virtualization of brain function and structure that could also raise the attention of any user interested in neuroscience. To give clinicians and interested users easy access to these tools, The Virtual Brain also provides a graphical user interface. The graphical user interface comes as stand-alone version for all major operating systems Windows, Mac OS and Linux and runs in a web browser. A monitor with a resolution of at least 1600 x 1000 pixel and a graphics card is recommended to ensure that the view is not truncated and WebGL visualizations are running accurately. In addition to the User Guide and online documentation for the GUI, which also includes some tutorials and information about data structures. TVB offers a mailing list as support platform for any open question or request. The User Guide is also embedded into the graphical user interface. Depending on the user's current location in the software, TVB has integrated a help overlay with the corresponding content from the User Guide.

[6]Electroencephalography, records electrical activity along the scalp – has a high temporal but low spatial resolution.

[7]Magnetoencephalography, records magnetic fields that are generated by electrical currents in the brain.

[8]functional Magnetic Resonance Imaging, measures activity of the brain by detecting associated changes in blood flow – like the *blood oxygen level dependent signal* (BOLD).

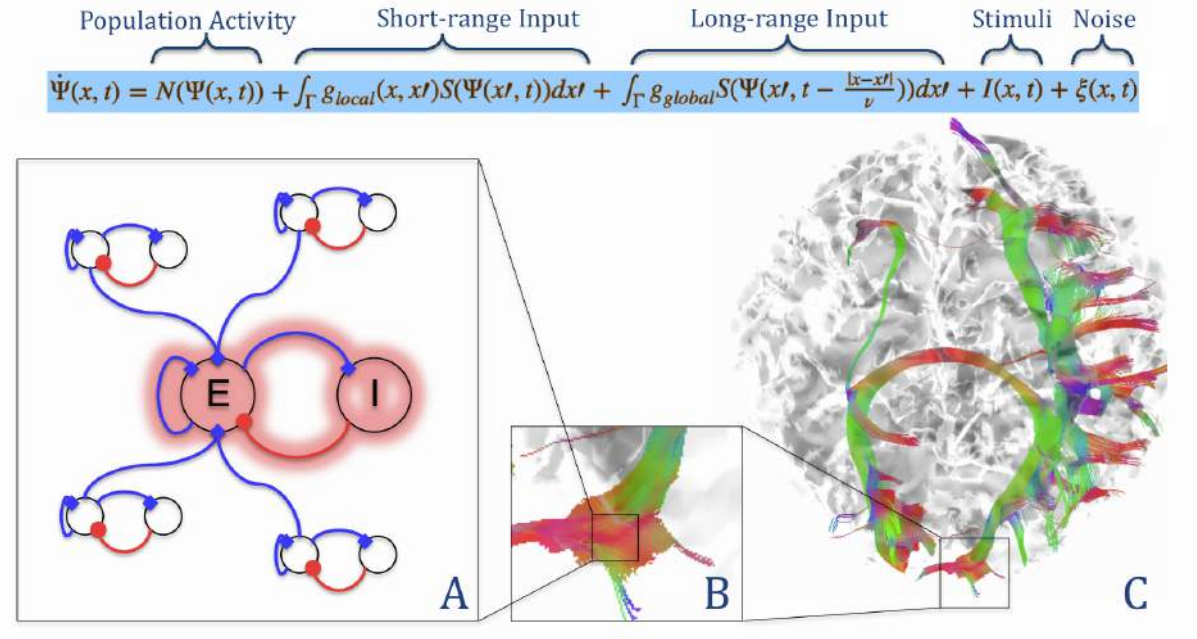

Figure 1.3: The Large-Scale Network Equation describes the spatiotemporal Network Dynamics in TVB [12]. It captures the activity of Neural Populations with its inhibitory and excitatory Neurons (A) up to the short-range (B) and long-range input (C).

One can find this help next to the heading in the upper section of the GUI. Outside the User Guide, some elements of the GUI are marked with a small question mark *"?"*. The user can find additional notes on specific elements by clicking these buttons. This can be quite helpful when the user is already experienced with the software or simulations, and only needs a quick reminder of used parameters, or other control elements, without disrupting the work session.

Contributing developers are welcome to enhance the application with new data analyzer tools or the integration of their dynamic models into TVB.

Target Group of TVB

One of the motivations for the development of TVB was to test hypotheses about structure-function relationships in the healthy or diseased brain for individual datasets. Therefore it was necessary to have a flexible system that allows the user to import their own structural connectivity datasets on a macroscopic scale, simulate mesoscopic brain dynamics with one of the mean-field models provided, and analyze and visualize the results [23]. Both interfaces, the script as well as the graphical user interface, offer the same set of functionalities and are independent from each other. The scripting interface offers more flexibility, but also expects scientific developers to elaborate everything on

their own. The users effectively have a wide spectrum of functionalities at hand in one tool.

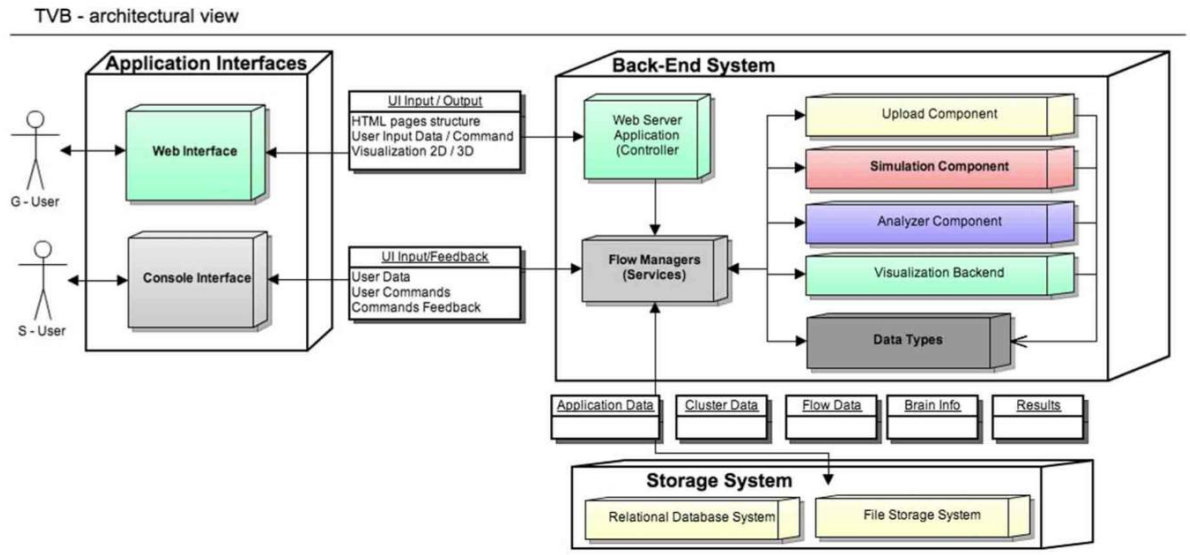

Figure 1.4: Excerpt of the architecture of TVB [23].

TVB Graphical User Interface

The Virtual Brain Graphical User Interface with all its functionalities is well structured and arranged, see Figure 1.5. The GUI is divided in three areas: Header menu (A), main area (B) and Footer menu (C). Starting at the top left corner the header menu has following elements: the number indicates the current project id (1) and the upside down triangle indicates running or finished operations like simulations or analyzer operations (2). If the image shows a grey triangle that means that there are no running operations. A blue triangle would indicate a running operation, a green triangle a finished operation, and a red triangle indicates a failed operation since the last time TVB was initialized. Next to the triangle is the button which will always open and show the submenu entries if available (3). In the project interface, as shown in Figure 1.5, all sub-menu entries are listed below the link, contained within the dark grey box (4). The name of the current active interface is written in the center of the header menu (5). With respect to the current section, TVB offers a link to the user guide containing some information about the functionality, and shot tutorials if applicable. In the upper right corner one can find an option menu for the current section (6). Especially in TVB-viewers or analyzers, one can change the color-coding, selected datasets or the frequency for TimeSeries playback.

In addition to the header area (A), the footer panel (C) is also permanently visible in the graphical user interface. While the footer menu stays unchanged with the links to all interfaces, the header menu and main area adapt to the content. Some viewers and

sections of TVB arrange the main area in two or more panels to visualize data and information within divided frames for efficient readability, see Figure 1.6.

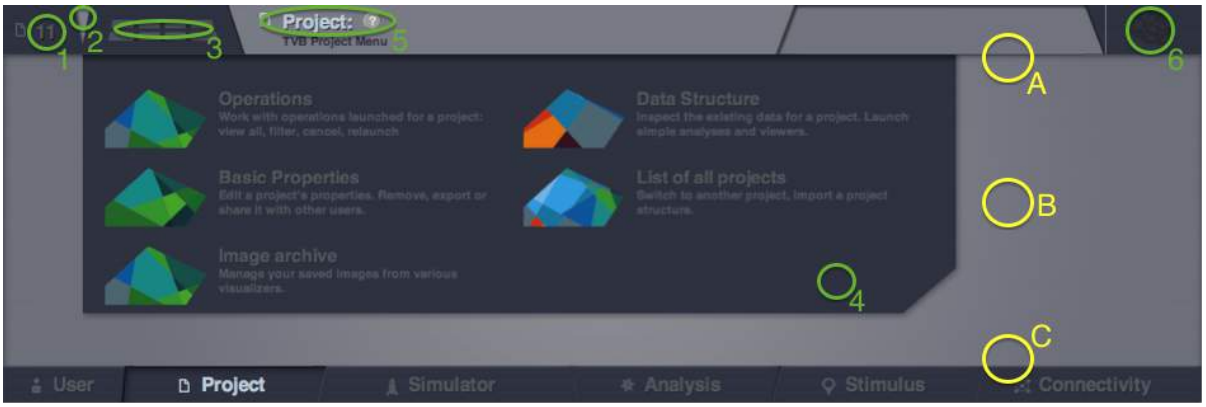

Figure 1.5: The Project Menu Interface of The Virtual Brain.

1.2 Motivation

TVB was first released on 13th of October 2012, for the Society for Neuroscience conference. The first release already included three different versions – a standalone version for single-users, a server version and a cluster version. It was presented with demo data for a connectivity dataset, cortical surface mesh and sensor position files. Since then many new features have been integrated. One can now manage datasets within a project by subject and import one's own data structures in various different file types, increasing flexibility and allowing the user to work and simulate with their own prepared datasets.

Even though it is much easier to get started with the graphical user interface of The Virtual Brain, new users still may lose track due to the unknown workflow and processes. TVB offers some help in form of written tutorials in the User Guide and within the GUI itself [26]. While the tutorials are reproducible, they do not help the user if they do not know what and how they are importing and simulating. TVB is open source and welcomes scientists to contribute to the project (TVB-Developer's-Reference 2014).

To facilitate usability we developed an educational tool – TVB-EduPack – that interactively assists the user to learn about functionality, handling and application of TVB. Due to the educational aspect of this software, TVB-EduPack is not only interesting for end-users, but also for instructors, since it also provides a scripting framework that allows the creation of helper functionalities, step-by-step tutorials and interactive demonstrations.

Figure 1.6: Screen splitting - The Connectivity Interface arranges the main area into two frames: the Viewer (F1) and a Control panel (F2).

1.2.1 Intelligent Software Agents

Difficultness of Working with TVB for Inexperienced Users

So-called intelligent software agents, respectively software assistants, can be real complex systems depending on the used soft- or hardware. The idea of such a system is to *personally* assist a user. Interaction is made possible through a wide range of scripted options and the use of methods from artificial intelligence, so that the system behaves "smart" [[18], [3]]. A famous intelligent software agent is Siri - the interactive help from the operating system iOS of the Apple iPhone that is integrated since iOS 5 (2011). The agent activates when the user asks questions to Siri instead of searching or using apps to get the information. Another example are event-related tooltips. One can find them e.g. in browsers (offering help, because the user is mainly surfing on German websites: *Do you want to translate this page?*), text-processing software (e.g. the paperclip in MS Office) or image processing software (e.g. suggestion when you are selecting a tool for the first time). The reason is obvious: the software is getting more complex, offering more or new features, but the user has to learn how to work with them. In the case of image processing or rendering software, the software is so complex that interactive guides or video tutorials are offered so that users get an introduction or first impression on how to start working.

When new users start TVB for the first time, they will see a very well structured and organized GUI, but in case that they don't know where to start or what they are able to do, they probably will get lost and frustrated.

Basic Helpers in TVB and Video Tutorials

TVB is distributed with a written User Guide that gives an overview about basic functionality and introduction tutorials for TVB. The User Guide is also integrated into TVB GUI: the user can open it in the different main sections (Project – Simulation – Analysis), jump directly to the pages and get some screenshots about the usage. Small tool tips also exist within TVB that inform you in brief sentences about the GUI elements – but these tips are not really helpful for non-scientific users.

1.2.2 Introducing The Virtual Brain - EduPack

A more Intuitive and Descriptive Tool was needed

Unfortunately, this kind of help was not enough because on the previous experience a more flexible tool was needed. The provided Helper elements of TVB are not sufficient for inexperienced users and created video tutorials do not offer much flexibility.

The idea for TVB-EduPack is similar to the described agents: offer information when help is needed, provide sample data and interactive tutorials to get started with TVB.

One major aspect is that TVB wants to welcome new users working with the software. These users can be advanced users, who are already comfortable working with some other simulation tool, or interested users that don't yet have the knowledge for working with this kind of software. Therefore we would like to present two different concepts: An *EduStart part*, were users shall get an interactive introduction to the software TVB while learning about concepts and methodologies in neuroscience. And an *EduCase part*, in which users will start working on specific concepts in neuroscience; like analyzing dynamic regimes and tuning parameters to generate neuronal activity with different models. Both concepts will be described in this thesis and part of the first TVB-EduPack prototype [17].

TVB-EduStart Tutorials

The first component of TVB-EduPack shall teach users how to work with The Virtual Brain software. The users will first build a foundation by getting comfortable with the basic functionality and usability, and then quickly move on to the tutorials. It will be possible to start tutorials on key elements within TVB. These tutorials shall guide users on the concepts and workflow of TVB, like how to create and manage a project, import datasets of different types for diverse reasons and how to use and manipulate them. This will be presented and integrated in TVB graphical user interface as step-by-step tutorials, which will wait for user interaction or as automatized demonstrations, depending on the content. Guided tutorials provide a lot of advantages. Users don't get distracted if they are working on a tutorial within the software, as opposed to having to read a separate manual. In the case of video tutorials, which may show the same information and actions on screen the user is working in, it is hard to follow along if the instructor does not match the background knowledge of the user. Then the user has to pause, track backwards,

and play again. For these reasons, the advantage of step-by-step tutorials where users can define their working speed on their own is apparent. In addition to tutorials on the functionality of TVB, it is easy to integrate related background knowledge on specific neuroscientific topics in the same format.

TVB-EduCase Tutorials

The objective of TVB-EduCases is to encourage users to start working with TVB. TVB offers a wide spectrum of functionalities to simulate and analyze data, but to get started on a better understanding about provided local dynamic models, TVB-EduPack will integrate features like pre-computed results. This will provide a handy tool for the users, allowing them to see exactly what they are going to compute. Pre-computed results also give a good overview for users who may want to understand the functionality of the software, but do no not have the experience to set up simulations on their own. The problem is two-sided: either users have a goal in mind and don't know how to get there, like simulating neuronal activity with a chosen local dynamic model, or users are experienced with a model and may need an overview about local dynamics and parameter ranges. Therefore, some use cases can be initiated or integrated via the xml-schemata, which will be explained in greater depth in chapter 4.1.2.

As an example of an TVB-EduCase, the paper *The Virtual Brain integrates computational modeling and multimodal neuroimaging* [21] presents data and results computed with the TVB framework. To increase comprehensibility and reproducibility of a research paper, TVB-EduPack could incorporate TVB-EduCases that are generated out of TVB Projects or by the authors themselves. This would allow the re-simulation of results by other researchers and open possibilities for further investigation of the data.

Improving the User Experience

The focus for TVB-EduPack will lie on the interactive components, and it is shown that interactive learning has a lot of advantages over written manuals and tutorials [19]. With TVB-EduPack we don't want to analyze the learning behavior of the users, but we want to offer the best chance to make progress in understanding the basic concepts and learning how to use them. Therefore, we do not only want to offer video tutorials on selected topics, but to integrate the user. It is shown that video lectures can present information in an appealing way, but if they lack of interactivity the users will not improve effectively [30]. Motivation is also a factor necessary for personal improvement. If the users get bored by excessive descriptions or frustrated by irreproducible tasks, they will move on – but without the usage of TVB. The step-by-step interactions should be easy-to-follow, balanced, dynamic and as challenging as feasible [22]. The higher the satisfaction, the higher the motivation and learning progress of the users [29].

Considering the wide array of possibilities within TVB-EduPack, the users will have the opportunity to choose what they want to learn along the way while being guided

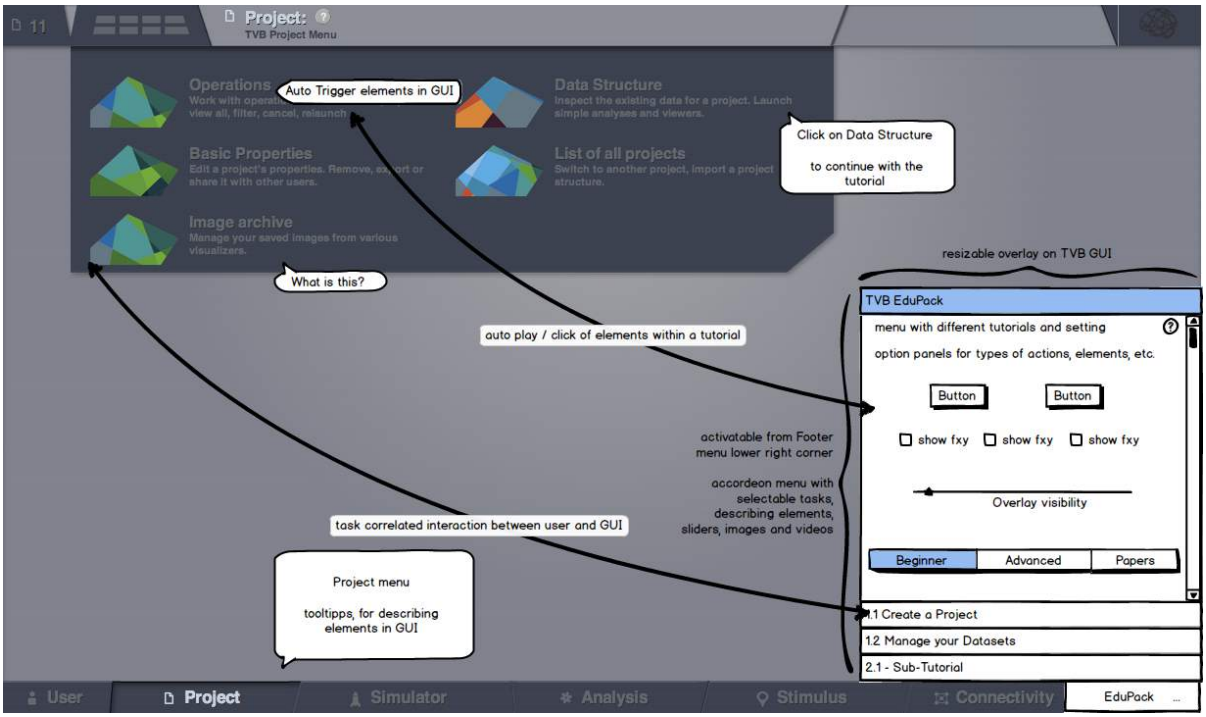

Figure 1.7: Mockup of a possible TVB-EduPack functionalities and the integration into TVB GUI (created with Balsamiq Mockups, 01/2014).

through the functionally and tutorials. Settings can be modified in the Control Panel within TVB-EduPack, and will allow the user to set up a desired configuration the tutorials and educational elements. But due to the different backgrounds of the users - some may be more interested in the human brain in general, and others may lack of knowledge on math or neuroscience and would like to improve in these fields – it shall be the users' choice whether and what they learn on a specific topic. And TVB-EduPack shall integrate, and make available these possibilities into the GUI of TVB – a draft of this concept is shown in Figure 1.7.

1.2.3 Summary

Designing and implementing TVB-EduPack so that it directly integrates into the graphical user interface of TVB is the first defined goal for this thesis.

Having tutorials that describe the GUI platform, how to work with TVB, and eventually use TVB for self-directed analyses, is the second goal for TVB-EduPack. Latter it shall be implemented with a generic tutorial interface that makes it possible to integrate the user interactively and work with single, flexible and combinable components to create tutorials.

2 Methodology and Design Process

"I like nonsense, it wakes up the brain cells. Fantasy is a necessary ingredient in living, It's a way of looking at life through the wrong end of a telescope. Which is what I do, And that enables you to laugh at life's realities."
Dr. Seuss

2.1 Engineering Requirements for TVB-EduPack

Some questions were crucial in respect to the engineering requirements: "What is the provided content?", "How shall the content get presented?", "What are the necessary interfaces in TVB to connect to the GUI?" and "What are the data flow constraints within TVB?".

The previous sections have described the generic concept. The following section will illustrate some technical requirements and requests for TVB-EduPack, which were developed by the project leaders of TVB[9].

- TVB GUI shall be the base for TVB-EduPack
- TVB-EduPack shall be an extension within TVB
- interactive courses are better than text and/or screen recordings
- tutorial sequences can be defined, and may have additional education components added by the users
- lectures about specific topics are to be followed by tasks for the users
- utilization of pre-computed results, simulations and TVB-projects
- target audience shall be those interested in neuroscience and advanced users that want to get acquainted with TVB

[9]Petra Ritter and other team members, http://www.thevirtualbrain.org/tvb/zwei/team

Whereas the first two points emphasize the importance of interfacing with the basic TVB functionality, the others focus on features for TVB-EduPack itself. The integration into TVB means that TVB-EduPack shall always be available for the user if requested. Consequently it needs to be integrated as some type of overlay, because it shall also cooperate with the different interfaces of TVB (Simulator, Viewer, Analyzer, etc.). A particular significant element should be the interactivity, probably implemented with a game-like approach and with the focus on serious learning. With the character of a computer game users would be motivated and challenged to solve new tasks or finish short quizzes (Kapp 2012). Collecting points and progress though completing levels should playfully motivate the user to continue working. The user should also derive motivation from the satisfaction of gaining insights and understanding from complex scenarios.

Advanced-User Aid

For the more experienced TVB users, the availability of pre-computed results would be the most interesting feature. With this integration users would be able to save computing time and reproduce simulations from others with less effort. The consolidation of this idea, and a scripting interface that allows you to generate own tutorials, will be shown in the following section.

2.1.1 Understanding the Workflow and Operation Mode of TVB

In order to design and implement the requested features it was essential to understand the basic concepts and workflow within TVB.

With the requirement of being user friendly and open for researchers or interested people, TVB comes with a Web Interface. The other application interfaces of TVB are not in this thesis, because the TVB-EduPack will be developed for the Web Interface. Depending on the TVB version, a webserver will run on the local machine (Stand-Alone version) or on a cluster to control the HTML interface. The interface consists of independent modules that are responsible for data reading (e.g. Data Reader), rendering (e.g. WebGL, Graph Renderer) and communication with the backend system (e.g. Command Adapter).

All pages and menus in the TVB GUI are generated through multiple HTML templates and filled by pyGenshi and JavaScript scripts at runtime. With respect to needed renderers, corresponding submenus etc. generic templates are filled with data from TVB-Backend to visualize the data. Each element of the generated pages has its own unique identifier, which is later important for TVB-EduPack to retrieve position, status and content to control or interact with the desired objects in the DOM-tree of the page.

The section in the middle of the GUI is reserved for the content of the selected interface. In that respect it would only make sense to integrate an interactive live assistant into all pages, because it shall work as link between the user and TVB. The result of that

should be an external interface, but integrated into the basic framework and graphical user interface of TVB. The different user groups are another requirement. Not all of them will work with or need TVB-EduPack, so the extension needs to be flexible and have the ability to be active and inactive. With respect to these two important requirements, TVB-EduPack needs to become a transparent feature that works in the current interface and allows the user to start and stop its functionality.

2.1.2 Integrating TVB-EduPack into the Graphical User Interface of TVB

TVB-EduPack needs to offer a comprehensible menu, where the user can set configurations, start tutorials or read and listen to instructions. Inspired by the submenu and option menu overlays within TVB, the only reasonable integration of TVB-EduPack was to create a new footer link, which deploys a TVB-EduPack overlay on the current page.

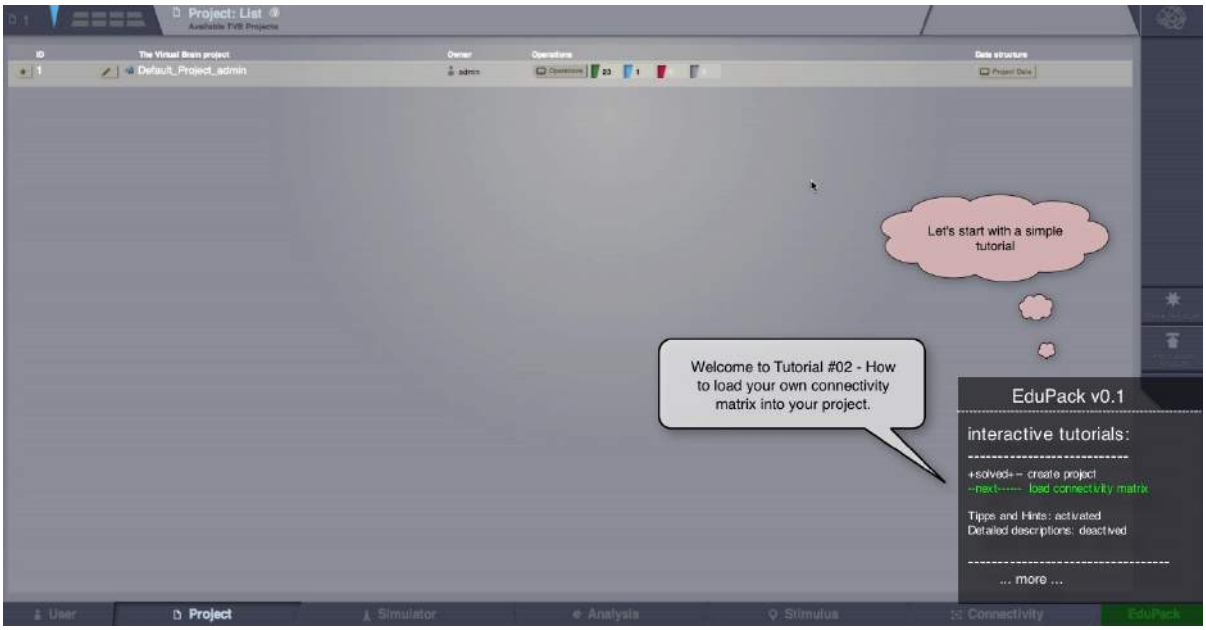

Figure 2.1: This screenshot is taken from a TVB video tutorial that presented the idea of TVB-EduPack (at the *Society for NeuroScience, 2013*).

As the draft in Figure 2.1 indicates, the TVB-EduPack overlay shall contain menu entries to start tutorials and options to configure the functionality. Another idea was to use the TVB-EduPack as helping agent like the paperclip in early versions of MS Office visualized here with the speech bubbles. They could be used to explain tasks to the user, but also react with help in situations when the user is lost in the graphical user interface. The speech bubbles could also highlight the task-related elements within the

TVB interface. The important factor is that the user gets guided and educated at the same time. Tutorials should be well structured, continuous and easy to follow. The more the user do on their own, the more they will learn in the end – and this is achieved when the user finds the path on their own, by clicking through elements, comes to understand the features and outcomes, and continues on with the next step.

2.1.3 Tutorial Requirements

Therefore it is important to integrate constitutive tutorials, helping the users from the beginning, but getting into more detail towards the end. For example, if a user is totally new to TVB, it is important that he knows where he can manage the data structures before he starts higher-level tutorials. It is crucial that the user stay motivated, so variety of clear instructions are important – images, animations and maybe questions at the end should always be used to support the tasks or descriptions. TVB-EduPack shall be the interface between user and The Virtual Brain; to motivate, educate and help when necessary. Mathematical problems especially can be a barrier for non-neuroscientists or researchers without scientific background. The goal is to offer help whenever the users need it – particularly when it leads to a better comprehension of the situation.

Usage of pre-computed Results

Because most of the neuroscience-interested users will work with the standalone version, TVB-EduPack should also include a feature, which utilizes pre-computed results. This would save computing time, and users could decide on their own if they want to simulate the data locally with their own connectivity data, or adapt selected parameters for in depth analyzes. Therefore it is necessary to corporate with the TVB-internal project management to facilitate an import and export of single simulations, results or whole project files into TVB-EduPack.

TVB-EduPack Framework

Given the mentioned requirements and the functionality of TVB itself, a generic framework named TVB-EduPack was developed that contains basic functionality and features to create and use an helping agent within TVB. This thesis will also present a scripting framework that integrates the implementation of TVB-EduPack into the graphical user interface so that every interested user can create tutorials and other TVB-EduPack related elements on their own and offer guidance for other TVB users.

2.1.4 Comparison against other Learning Tools

Many distributors of more complex software systems, like *MathWorks Matlab* or *Adobe Photoshop*, only offer written tutorials and manuals or screen recorded video tutorials on

selected topics. The only interactive support is from the active community exchanging user-to-user, and user-to-developer, assistance in forums. As described before, TVB-EduPack shall become a community driven project, and since TVB continues to be developed (and therefore changing in graphical user interface, too), video tutorials are not a favorable instrument. What we can learn and take from these videos are the structures and presentation style. Another key element found during research is that most software distributors use screen recordings, or at least the style of screen recordings, for the presentation of the software and all its key components. Due to the fact that TVB is free and everyone can download it, one could also integrate a software presentation tool for TVB with the help of some TVB-EduPack components. Just like other companies demonstrate the features on slides, flyers and videos – TVB GUI could represent itself through automatic control from a script.

2.1.5 Summary

For the development of the interactive helping platform, it was necessary to weigh and order all the desired features in order to select a basic set to be implemented within this thesis:

- develop an assistant that guides (e.g. features and functionality of TVB) and educates (e.g. about neuroscientific methods and concepts) – called TVB-EduPack
- integrate configuration panel for user options within a menu to allow adaptions for every user specific to their needs and qualification
- use motivating, game-like approaches to various problems (mathematical and neuroscientific topics)
- develop TVB-EduStart and TVB-EduCase tutorials on specific sections: featuring detailed descriptions of what, why and how it's done in TVB
  - step-by-step tutorials on functionality, neuroscientific topics
  - different helper tools are needed: highlighting, comparison, text, overlays, animations
- design and implement TVB-EduCase library to store and integrate pre-computed datasets, and to generate a quick overview of TVB's dynamic repertoire for chosen parameter settings

Based on the requirements, a flexible tool is needed that allows both the possibility to generate tutorials with interactive elements, and the feasibility to integrate them in the existing TVB graphical user interface. To offer maximum flexibility between user and TVB, the tools need to interface directly with The Virtual Brain. Due to this constraint, the utilization of other tools, which offer scripting frameworks are not an option. The evaluation of elements, steps or tasks within the scripts needs to be executed inside The

Virtual Brain. Therefore it is necessary to develop directly in TVB to allow interaction at runtime, based on predefined functions and options from a scripting framework.

2.2 Design Concept - Two-Way Interaction

The central idea of TVB-EduPack is the functionality as an interface between the user and TVB itself – the so-called *TVB — TVB-EduPack — User* circle. In this position it shall help the user to learn to work with TVB. Therefore it needs to interact with the users and TVB as well.

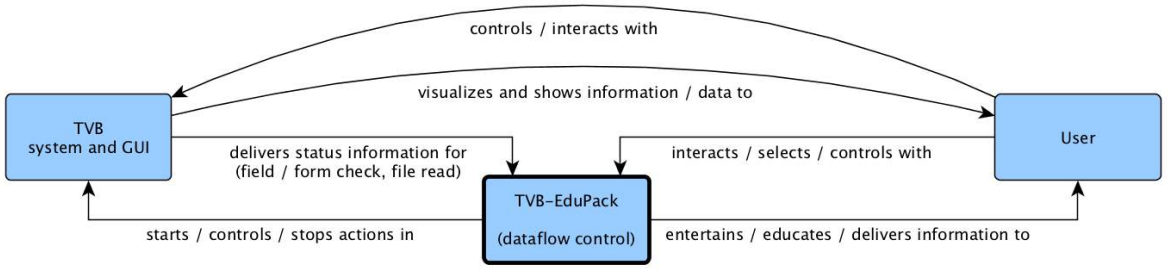

Figure 2.2: Two-Way Interaction of TVB-EduPack with User and TVB, the TVB — TVB-EduPack — User Circle.

But as described earlier – TVB-EduPack will only be an addon that the user can activate and deactivate as he or she wishes. Figure 2.2 shows that the interaction between The Virtual Brain and the user is not affected. Users can still manage, control or interact with TVB without getting disrupted with tips, pre-computed results etc. The latest TVB version functions as before, with the same content, except that there will be a new link to TVB-EduPack in the lower right of the footer menu. As long as the user does not click the link, TVB-EduPack is deactivated and not working in the background. Figure 2.2 also shows that the information flow or activity is not unidirectional but bidirectional between both User and TVB-EduPack and TVB-EduPack and TVB.

With TVB-EduPack as the active interface between user and TVB, it will receive information although the user may be interacting with TVB. These status information updates are necessary for TVB-EduPack to manage the internal dataflow. That means that not all tutorials or sections can and will be available all the time, or an already selected tutorial or EduCase is awaiting interaction between user and TVB, while TVB-EduPack shall receive this information automatically from TVB.

Another example is the functionality of an auto-check of parameter values in some form fields, e.g. on the simulator page. When the user is requested to set up a project

with essential settings or starting a simulation with requested parameter ranges, TVB-EduPack will check the data live so that the user is limited in the number of actions within TVB – because he or she hast to adapt these values first. But TVB-EduPack can also directly interact with TVB and send commands (on requests) to start, stop or control actions inside TVB.

TVB-EduPack will be acting as overlay in the lower right corner of the browser window, covering some small area of the original TVB window. The interaction between user and TVB-EduPack takes place in this overlay: selecting tasks, menus or starting actions that are then sent to TVB. To offer additional help, there will also be pop-up windows and Helper elements in the browser containing exercises, information, animations or figures.

Comparison against a non-Interactive Tool

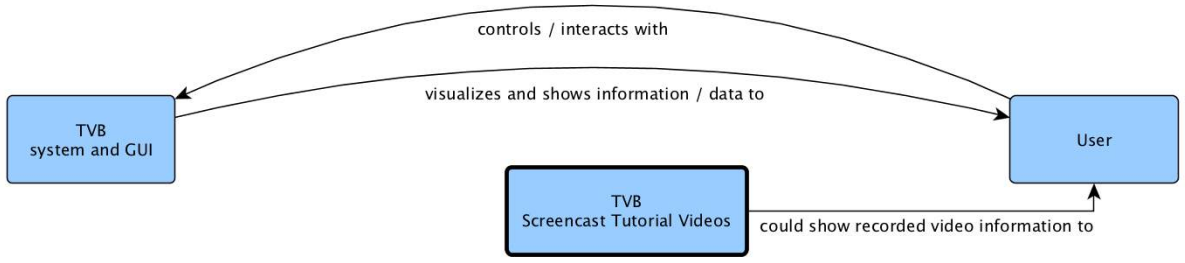

Figure 2.3: A non-interactive tool like video tutorials would not be able to incorporate both user and TVB.

As Figure 2.3 clearly shows – If the module only displayed video tutorials, there would be no interaction or advantages for the user. The only interaction available to the user would be to pause the playback, then follow the instructions within TVB. Even though such a feature would be more appealing than a written tutorial, it is still missing all the advantages of interactive learning.

2.3 A Flexible XML Approach

In this work I present an approach, which makes it possible to integrate interactive tutorials into the TVB system, to educate users that are willing to learn topics from neuroscience, and also how to work with TVB. With the implementation of a XML scripting interface, users will be able to create tutorials on topics concerning TVB, their research, or something completely different, by using predefined actions and elements [20].

In addition to the learning functionality, TVB-EduPack shall also provide a teaching feature. It was be possible to write all the tutorials just as list of JavaScript commands but then it would be difficult to incorporate new steps in the sequence and would imply too much programming time. Finding this direction unsuitable for the purposes of TVB-EduPack, we looked to Multimedia-Development tools like *Adobe Director*, *Adobe Flash* and many others, as they incorporate a scripting language to offer more flexibility than just a fixed sequence.

We decided to provide TVB-EduPack with an XML interface for everyone who wants to create and prepare tutorials for the community. Elements, tasks and interactions can be included and defined for customized tutorials or simulations, so that everyone is able to recreate specific tasks that need additional descriptions or information. The intention is to take advantage of the defined TVB-EduPack XML schemes by providing a tool that facilitates users to share their knowledge the growing community.

Similar Scripting Tools

The discussed features and requirements lead to a customizable language that can be applied by other users to create tutorials or EduCases[10]. Custom defined languages that perform some processing after getting executed or compiled are common. One example is *Apache Ant*[11], which is based on XML to generate and automate software build processes. It is implemented in *Java*, requires *Java* and is mainly used to build Java projects. A more applicable project would be *JSP*[12] – a technology, which allows you to create dynamic elements for web-based applications. It is also written in Java and comes with the easy to use benefits of *Java*. The platform independence and possible integration of *Java* in Web pages would make it a very powerful and dynamic tool to control TVB. The same applies for *Jelly*[13], another tool for users to create custom actions using an XML based processing engine.

Concerning these tools, none may be considered for the implementation of TVB-EduPack because it shall get integrated as a plugin for the existing TVB framework, without including third-party software dependencies, such as Java or other frameworks. However, the concept of using XML as custom defined language to perform some action at runtime will be an important part of TVB-EduPack.

2.3.1 XML Structure Requirements

The key concept for designing and implementing the TVB-EduPack consists of two fundamental elements:

[10] I will later refer to the differences of *normal* tutorials and the characteristics of an *EduCase*
[11] http://ant.apache.org/
[12] http://www.oracle.com/technetwork/java/javaee/jsp/index.html
[13] http://commons.apache.org/proper/commons-jelly/overview.html

- generic XML schemes for the creation of TVB-EduCases
- integration and usage of generic TVB-EduCases in TVB

Originally the idea was integrating simple tutorials into TVB. Therefore elementary actions were developed that would be helpful for the user.

2.3.2 Different End-User Types

With respect to different backgrounds and experiences, TVB-EduPack will focus on three End-User groups:

- Basic Users - that want to learn about how to operate TVB, will probably work with TVB-EduStart tutorials.
- Intermediate Users - that want to work with TVB for their own studies and want to learn about neuroscientific topics, will rather work with the provided TVB-EduCases.
- Expert Users and Programmers – that would like to share their knowledge and experience with the community of TVB, will use the presented XML scripting interface of TVB-EduPack.

2.3.3 Guided Tutorials for Basic and Intermediate Users

Through the XML scripting interface it is possible to create TVB-EduStart or TVB-EduCase tutorials. There are no big differences between these two forms. The origin of the definition comes simply from the words "*Education*", "*Start*" and "*Use-Case*" – in short: EduStart and EduCase. Use cases strictly follow a routine in TVB – either for beginners or more advanced users. For example, for uploading different file types - one has to follow a few steps from file selection up to the confirmation of a successful upload. Whereas for the beginners the simple workflow is important, the advanced users have more specific needs. But in general the term "tutorials" is used in any case within this thesis, which describes functionality of TVB-EduPack or TVB, including Helper Elements, fragments of a TVB-EduCase or even additional materials. See section 3.2 and following.

A TVB-EduCase refers to one use case in TVB, so it may consist of:

- one or more **preconditions** for an use case, step or sub-step
- one or more **steps** and **sub-steps** which describe and guide the use case
- one or more defined **tasks** that need to be fulfilled within a step or sub-step before being able to continue to the next one

- one or more assigned **actions** to every step or sub-step

To allow a flexible implementation of use-cases, the above-mentioned elements need to be independent among them. The preconditions for example are an important feature when TVB-EduPack wants to check "Is the user even ready for this use case?" or, "Does the user possess the constitutive knowledge?". If the preconditions are not yet fulfilled, the user shall not be able to start this use case or continue with the next step. The aim is to help the users, especially when they get stuck or frustrated within parts of the tutorial that were too advanced for them.

The break down of a TVB-EduCase in steps and sub steps is a design decision. It improves the overview and divides the TVB-EduCase in several independent, constitutive parts. This will be particularly helpful when the users shall focus on their tasks and not get distracted by previous or upcoming content.

Within these steps and sub steps, the users will either find some written descriptions, images, and even tasks - like hitting a button to continue, or to look at a form field in order to gain more knowledge about a selected parameter. The tasks and general content will be supported with graphics, highlighted elements, and window overlays to call the users attention to selected elements.

There are many possibilities for the combination of tasks with actions, and preconditions for the next task respectively applying constraints on the current task. More details will be provided in the implementation section with examples of a TVB-EduCase, including XML structure and task content in the practical section, chapter 4.1.

2.3.4 Expert Users: Tutorial Creation

The integrated features that can be used of the XML scripting interface will get documented into a User Guide to allow other users to create tutorials, TVB-EduCases or automated operations on their own. In the future it will be desirable to have a modular design principle for the user to click, drag and drop elements within an editor. By now we will consider manually written tutorials that can be integrated into TVB-EduPack.

2.3.5 XML Feature - Automated Application Control

The developed XML scripting interface allows the user not only to create tutorials for users, but also to generate interactive steps that do not wait for a specific input like hitting a button or setting a fixed value into a form field but does that automatically and only needs a simple input on the keyboard to trigger the program to continue. This could be useful for all target groups, and can be seen as a presentation tool. The idea for this feature developed while showing some of the main features of TVB to new users and colleagues.

The automated application control will use the same set of actions that are also used for the tutorials, but it will also need some additional features for implementing the automated operation mode. Most of the actions used within the tutorials focus on highlighting elements or waiting for some input from the user. The big difference between automatic and user interactive operation modes is the attention level of the user. When the users are asked to do something, they control the system with the mouse, check the graphical user interface and continue when everything that they were asked to do is done. The actual difficulty when it comes to an automatic operation mode is to keep the user's attention. Therefore some additional mouse and click effects are needed so that the users can see where and what happens on the screen in order to not get lost. It is possible to observe this during video lectures or live workshops where users have problems following the instructions.

Despite this, these automated instructions can still be a useful tool for the users. Attractive transitions and animations can be a key element to help users follow the course of actions. Users will still be in charge of the pace, because this can be triggered through the keyboard like in a *Microsoft PowerPoint* presentation. An input can then start e.g. a sequence of transitions, animations of mouse events or form field changes.

The automated application control can be also useful when a user is stuck somewhere in the GUI. TVB-EduCases can make use of a help function that activates when a predefined amount of time has passed without user input. This will not be part of the first TVB-EduPack prototype, but is a high priority consideration on the To-Do list.

2.4 Design Approach - Summary

Having sorted all requested features, there are three main components that need to be developed and integrated. First is the integration of TVB-EduPack itself. Integrated into the panel of TVB, TVB-EduPack will pop up with a click and show the user the main menu, options and selectable tutorials to get started. The second work package will be the development of a generic XML scheme that allows integrating and creating these tutorials. Additionally there is also a counter part – package 3 – a program that reads, interprets and executes those XML scripts within TVB. Otherwise there would be no interaction, guidance nor extra information at all.

This package will be the most important contribution of this thesis, especially because it shall get developed in a way that it could be easily extended with more features in the future. All packages will be developed with the idea of two-way-interaction, considering TVB-EduPack as a link between the User and The Virtual Brain.

3 Description and Implementation

"Tell me and I forget, teach me and I may remember, involve me and I learn."
Benjamin Franklin

The following chapter will describe the implemented packages in detail. It will deal with the design, the different XML schemes, all necessary XML structure elements, action primitives, the evaluation of XML scripts with JavaScript and the way of integrating pre-computed results into TVB-EduCase library. The scope of this project is to develop a first prototype of TVB-EduPack, with a basic set of functionality that can be extended with new action primitives and features in the future. All features of the first prototype were implemented and tested with the latest available version of The Virtual Brain 1.2, September 2014.

3.1 Package 1 – Representation and Integration of TVB-EduPack

TVB-EduPack seamlessly integrates into the software architecture of TVB as an additional module. The rendered TVB Web Interface is extended by a TVB-EduPack overlay that is included into the lower right corner. Therefore the *footer.html* template and the correlated CSS design file of the footer menu were adapted. The Virtual Brain workflow is not affected when TVB-EduPack is inactive.

Figure 3.1: Detail of the TVB footer menu with TVB-EduPack in the right corner.

3.1.1 Integration into The Virtual Brain

To complete the integration of TVB-EduPack into TVB, it was essential to import the implemented script (*main_edupack.js*) and design file (*edupack.css*) into the

base_template.html. The *base_template.html* of TVB is the base structure element for all generated and rendered pages within TVB. It assembles the menus, tabs and visualizers as requested from the related pages. With the integration of the two TVB-EduPack related files, the XML-based TVB-EduPack menu will always be loaded in the background of the application. This will lead to no further delays when TVB-EduPack iss activated. It is not possible to start TVB-EduPack in the Login and User Configuration page, because these two pages are not generated from the *base_template.html* file.

3.1.2 HTML Templates

At the time the user opens and starts TVB-EduPack for the first time, related preference variables will be created and saved in the background. For this purpose the *HTML5* feature *localStorage* is used. It facilitates to save data locally in the browser on the client site without using cookies or sending data to a server. This feature is used to save name-value pairs of preference information like tutorial progress and user related information. Unlike Cookies, which would be loaded at every server request, the use of the *localStorage* has the advantage that data is only loaded when it is requested. Furthermore, it is also possible to save bigger datasets. The limitation is set by the browsers (*Google Chrome*, *Opera* and *Mozilla Firefox* support 5 MB, *Microsoft Internet Explorer* 10 MB,(Hickson 2013)).

Using the method calls *localStorage.getItem(name)*, *setItem(name, value)* and *removeItem(name)*, facilitate tracking of local user data while they are working with TVB-EduPack. The last saved configuration enables the user to continue after a page refreshed or if TVB is restarted for another reason. This function is also used to save and load user data about the activated or deactivated TVB-EduPack. In the case that TVB-EduPack is active, more configurations of visible Helper elements, Tutorial or Menu window positions are loaded. The tools work with the structure level information and IDs that are forwarded from the XML scripts. For more details about the structure levels and attributes of the XML scripts, see section 3.2.

3.1.3 CSS Components

Cascading Style Sheets (*CSS*) are common usage for the uniform design and functionality of specific Web elements. The whole TVB Web GUI is implemented and composed with the usage of CSS and so is TVB-EduPack.

The GUI of the activated TVB-EduPack is a * * container that is in the foreground of TVB GUI. The container consists of a window overlay with attached and partially transparent images in a colorful TVB style. To accentuate and show simultaneously the relation between TVB and TVB-EduPack, we used TVB colors but chose a hand drawn style. This design concept of images and animations is recurring through all TVB-EduPack elements that have been prepared and implemented for the first prototype. The

size of the overlay rescales with the browser size and is slightly transparent so that the user still sees the TVB GUI.

Figure 3.2: The screenshot shows a section of TVB with activated TVB-EduPack. It presents a prototype of TVB-EduPack with border images, selectable Helper elements for TVB GUI (and a slider for their transparency) as well as a structured menu, generated from an XML tutorial file.

3.1.4 Visual Design of TVB-EduPack

Figure 3.2 shows a capture of an activated TVB-EduPack overlay, which was taken in the Connectivity page of TVB. The green elements illustrate the generated tutorial menu. The example consists of four elements – *Step_00* to *Step_03* and has the title *Sample tutorial*. The description of the tutorial is integrated below the checkboxes of the TVB-EduPack Helper elements.

The size of the TVB-EduPack overlay box is fixed and if more vertical space is needed, a scrollbar will appear. This happens quite often due to the chosen accordion design of the menu. When the user opens an element from the menu, the content will be released out of the clicked element and the other elements will slide down. Furthermore, it can happen that the overlay window of TVB-EduPack lies on top of tutorial related form fields or buttons. If this is the case, TVB-EduPack can be shifted to another position

by using an action primitive that moves the overlay. It was also considered to make the whole TVB-EduPack movable in the GUI. But this idea was discarded to avoid unnecessary distraction. For more information on action primitives, see chapter 3.2.6.

Organization of TVB-EduPack Elements

All design elements of the menu are defined in CSS groups and come to life with corresponding JavaScript functions. A tutorial or an integrated menu shall consist of three structure levels. The first level is thought as a grouping element for one or more smaller steps. These smaller steps are called sub-steps and can be regarded as the second structure level. They should contain an instruction for the user and optional action primitives, constraints or other featured elements that will be described in section 3.2.3. If one level is missing or badly integrated, errors may occur either in the animations of the menu or the menu may not be consistent and working appropriately.

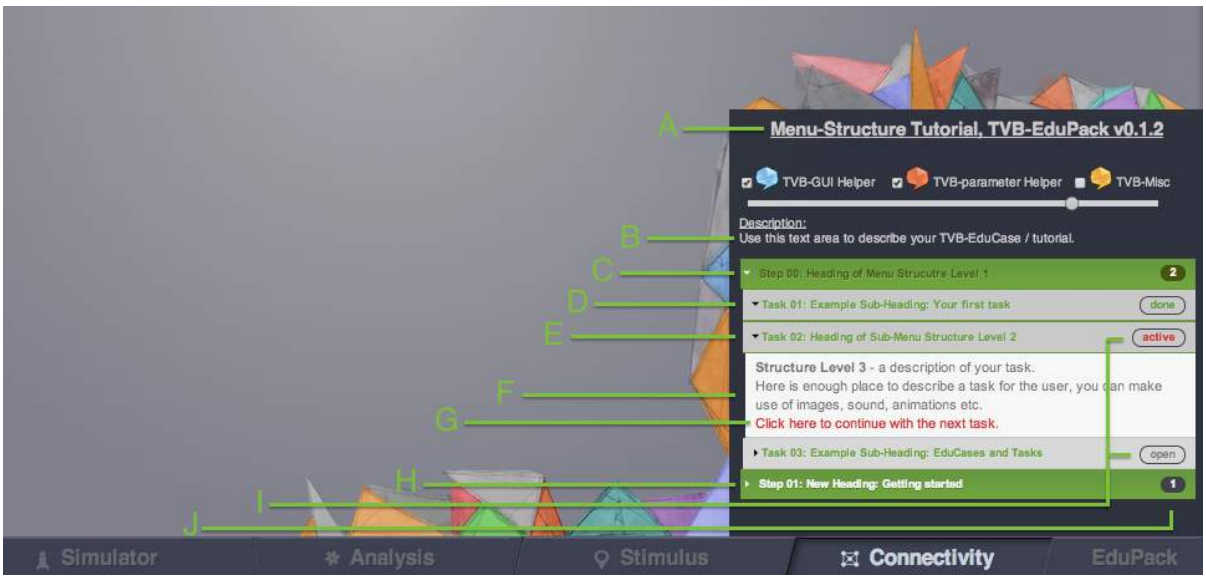

Figure 3.3: Description of TVB-EduPack: the numbered elements show all visual and content related elements of an activated TVB-EduCase.

A The Title of a Tutorial and an automatically appended internal Version Number of TVB-EduPack.

B Description Text for the EduCase that introduces into the Topic.

C Menu Structure Level 1, a <*step*> element encloses also a <*step-id*> (here starting at *00*), a title element <*label*> and all <*sub-step*> elements that shall belong to this structure level. In the figure the element is opened and shows three incorporated <*sub-step*> elements.

D A closed and finished <*sub-step*> element that contains a green heading within <*task*> brackets and a <substep-id>. The heading color relates to the <*type*> of the <*sub-step*> - this may be a helpful feature for the user because TVB-Helper elements follow the same principle. One can choose between description (*green*), parameter (*orange/red*) and misc (*yellow*).

E This is an example of an active <*sub-step*> element. Just like in *D*, it belongs to Menu Structure Level 2 and the described <*step*> element in *C*.

F Structure Level 3 contains the written Task between <*detail*> brackets in the corresponding <*sub-step*> element. It can contain HTML Style elements, images or integrated video components.

G A <*sub-step*> element can have an auto-finish constraint, or as shown here – contain a manual link that will mark the element as finished after clicking it.

H This is an untouched new <*step*> element with the ID 01.

I After clicking the link in G, the status on the right side will change to *done*. When a <*sub-step*> element is clicked, all related action primitives will be loaded and stay active as long as they are not fulfilled or the <*sub-step*> element is finished or closed manually. When the user achieved all constraints, the element will change to *done*. These elements can also get the status *open* (not clicked or solved yet) or *blocked* (when some predefined constraints are not fulfilled yet).

J While the status elements of <*sub-steps*> indicate the finished, blocked or open elements, the status element here shows the number of the contained and still open <*sub-steps*>. For <*step-id*> 00 it shows two unfinished <*sub-steps*> (<*sub-step*> element 01 is already marked as done) and <*step-id*> 01 either contains only 1 sub-step element or some were already finished at an earlier moment (this is not visible here).

Figure 3.3 shows and describes the visual result of the JavaScript and HTML template evaluation of a simple XML tutorial. All used XML tags use opening and closing tags: <*tag*> *content* </*tag*>. An XML tutorial follows a strict hierarchy structure. <*Step*> elements enclose their properties as well as all <*sub-step*> elements enclose their properties, featured <*detail*> text, <*action*> primitives and different <constraints>. This hierarchy structure will be featured in section 3.2.5.

3.2 Package 2 – The XML Schemes

For the first prototype of TVB-EduPack there are two different types of XML schemes in usage. One is for the tutorials respectively TVB-EduStart / TVB-EduCases and another one for so-called Helper elements. Both schemes are independent from another and can be created or adapted to aim at different user groups and integrated into TVB-EduPack.

3.2.1 XML Helper Elements

The graphical user interface of The Virtual Brain offers a helper element that is visualized with a small question mark, see Figure 3.4. There is nothing wrong about these helper elements, but they are only offered at form fields and contain some written text from the TVB-API. Taking advantage of this idea, I developed an XML schema for TVB-EduPack Helper elements.

The TVB-EduPack Helper elements shall be an additional tool for new users to get help from the graphical user interface itself. There are so many buttons and elements on screen that are unclear for users, especially in the beginning, so that their integration can be really helpful.

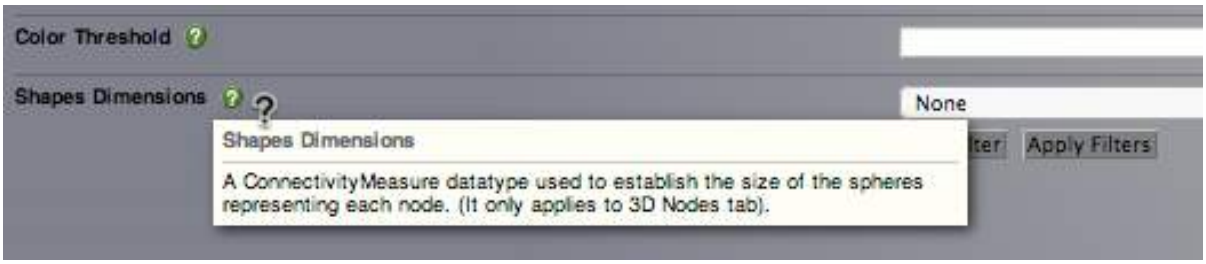

Figure 3.4: The integrated TVB helper tool - here describing a form field in the large-scale connectivity menu.

Adaptable Configuration for TVB Helper Elements

To accomodate users with different experience working with TVB-EduPack, I developed a tool that distinguishes three different types of Helper elements. The three types differ in color and correlate either to TVB-GUI elements (*blue*), parameter related in-depth knowledge (*red / orange*) or miscellaneous information (*yellow*).

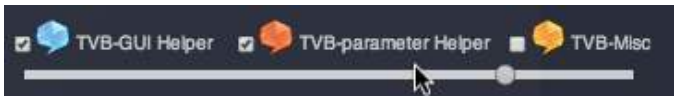

Figure 3.5: The three TVB-EduPack Helper elements can be selected from TVB-EduPack menu. The slider can be used to adapt their opacity level within TVB GUI.

It is up to the user to decide which elements of the loaded Description XML File are shown. He or she can select any number of groups and even adapt the opacity of the elements. The latter can be useful when these elements hide other objects on screen, which is due to the limited space on screen. When the user opens a TVB-EduPack Helper Element, the main screen of TVB will fade out with a dark gray layer and a speech bubble will appear that reveals the helpful content. The size of the speech bubble overlay relates to the content and is opened directly next to the object. The overlay is movable, for in the event it obscures other object in TVB's GUI.

The authors and editors of the elements are welcome to use all kind of media elements. HTML tags can be used to emphasize text, integrate images, diagrams or even link to local stored videos or clips on the web. Figure 3.6 and Figure 3.7 show an integrated example in TVB. The element will close when the user clicks on the "x" in the upper right corner or into the shaded background. The behavior and visuals work with its own CSS class and JavaScript so that they can be adapted in style and functionality.

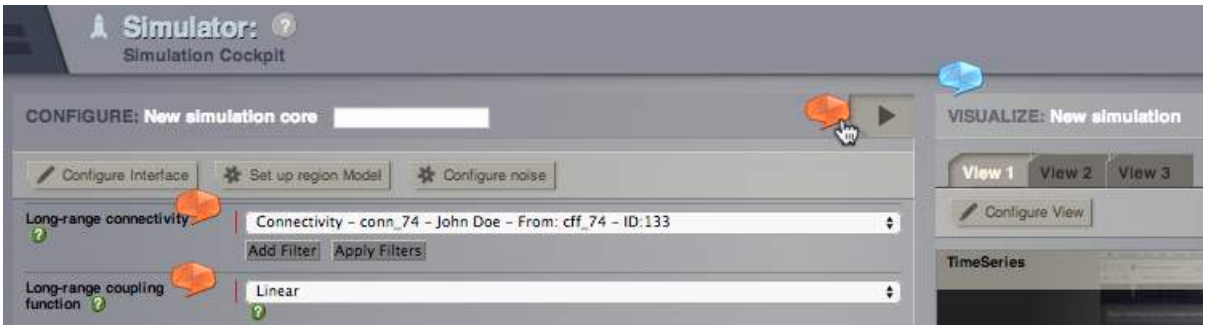

Figure 3.6: Excerpt of TVB Simulator page with integrated parameter Helper elements (red / orange) and TVB-GUI Helper element (blue).

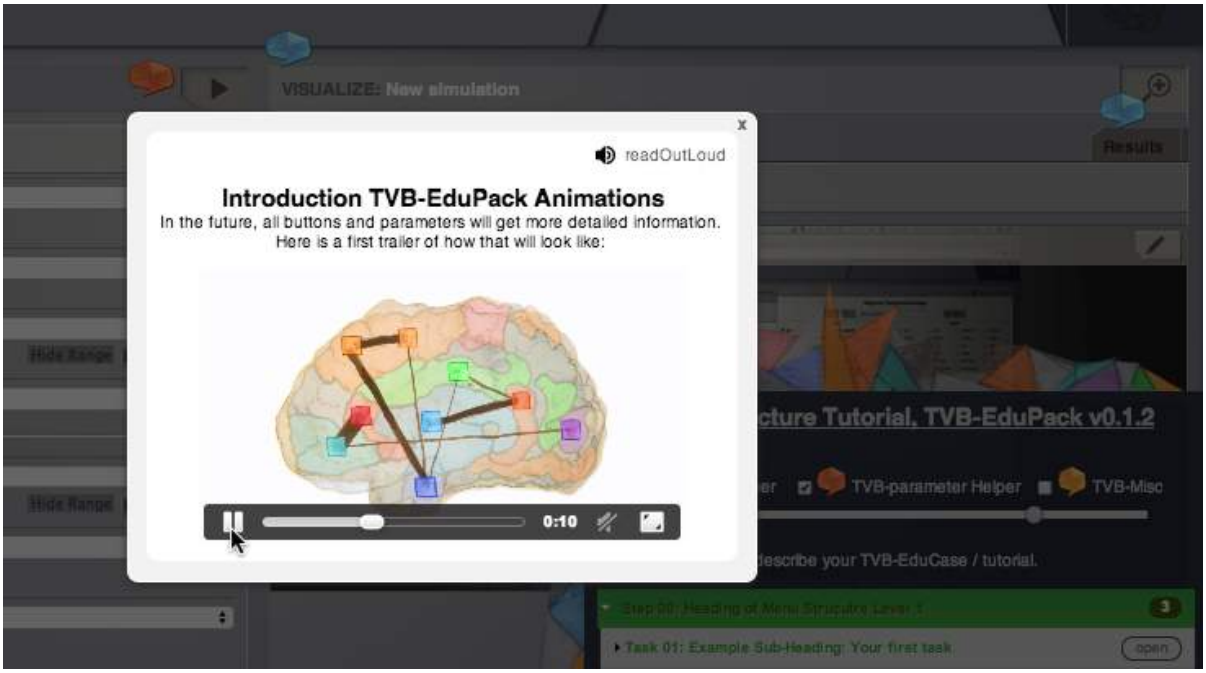

Figure 3.7: The <overlay> shows the content of a parameter Helper element with featured text, read out loud functionality and an integrated animation. The correlated code is listed in Listing 3.1.

3.2.2 Structure for XML Helper Elements

The format of the XML scheme is quite simple. In an enclosing *<edupack_xml_detail_description>* tag all desired elements can be defined by using the *<description>* tag as shown in Figure 3.8. Between the start and end tag of *<description>* the author can write the correlated text. It is recommended to work with the enclosing command *<![CDATA[... content...]]>* around the text. Otherwise it is not possible to integrate HTML tags and other errors will occur when using special characters. The related code fragment of the presented overlay from Figure 3.7 is listed in Listing 3.1.

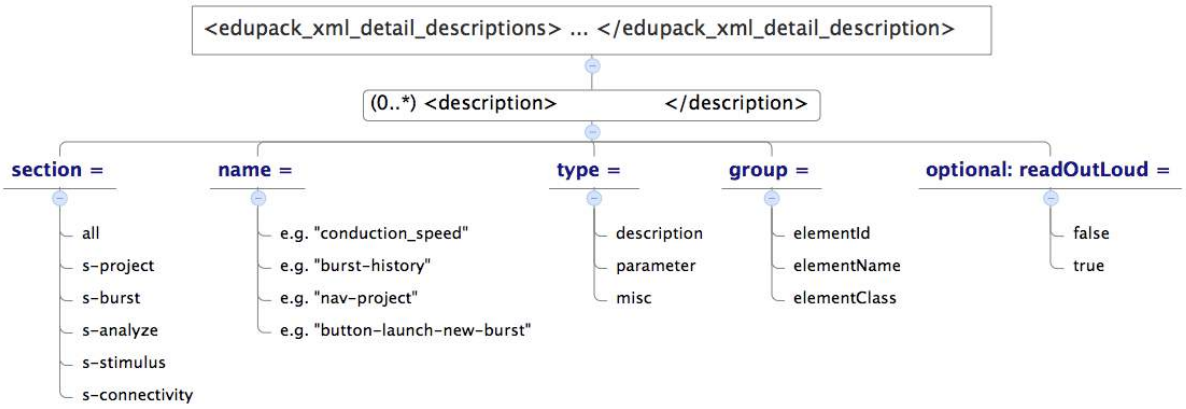

Figure 3.8: XML structure with necessary attributes of a Helper element.

Listing 3.1: Source Code of an implemented Helper Element. The Visual Result can be seen in Figure 3.7.

```
<description section="s-burst" name="button-launch-new-burst" group="elementId"
            type="parameter" read="true">
    <![CDATA[
    <h2>Introduction TVB-EduPack Animations</h2>
      In the future, all buttons and parameters will get more detailed information. <br>
      Here is a first trailer of how that will look like: <br><br>
      <video id=video width=800 controls preload=auto autoplay>
             <source src=/edupack/videos/_renders/intro_v3_720.mov >
      </video>
    ]]>
</description>
```

TVB Variable Names
The difficulty in creating TVB-EduPack Helper elements is that the author has to know about TVB internal DOM-Tree[14] IDs of the elements (Hégaret 2005). Information about element- or class names would also be sufficient. Without the information it is not possible to create the Helper elements in the GUI of TVB. However, I created an example XML file for almost all elements of TVB. The description tags of these elements are commented out and can be integrated and filled with one's own content as wished.

Evaluating and retrieving the position for DOM-Tree elements that did not have a specific id was difficult. There are still problems when a user works with two monitors and resizes the browser window. Usually one can fetch the resizing event from the window DOM object, but this was not working to satisfaction (tested with *MacOS* and *Google Chrome*, *Safari* and *Mozilla Firefox*). The repositioning and visibility of Helper elements is also required when the user scrolls, like in the TVB simulator page while setting the parameters.

3.2.3 Structure for XML TVB-EduStart and TVB-EduCase Tutorials

As described in section 2.2, TVB-EduPack can be seen as a link between the user and The Virtual Brain. With the Two-Way-Interaction we want the user to work with TVB, while getting help and guidance instructions from TVB-EduPack when it is necessary. According to that, the user learns how to control TVB and simultaneously TVB-EduPack interacts with the user and the graphical user interface of TVB.

Putting these objectives into practice was only possible with a generic concept. In the beginning some single use case scenarios were hardcoded to test how elements of TVB could work and interact with the system and user. Although they briefly fulfilled the requirements, the usefulness and functionality was very restricted due to a linear process order. With the requirement to allow people to create their own tutorials, there was only one possible solution - to develop an event-oriented modular system that is easy to handle.

3.2.4 Event-Oriented, Modular Concept

The Virtual Brain is a really powerful neuroscientific software that allows the user to import, manage and process neuroimaging datasets as well as manipulate and use the data for further simulations and analysis. There are no strict linear process orders or limitations on how to work with TVB. The same should apply for TVB-EduPack: no strict

[14]The HTML DOM (Document Object Model) contains all HTML elements from a webpage as objects in a tree structure. With the help of the DOM one is able to get, add, delete or change HTML elements and their corresponding properties, methods and events.

rules, only guidelines – the user shall still be allowed to make their own decisions and experiences with TVB. Therefore the tutorials and TVB-EduCases have to be flexible and need generic concepts to be useful in practice.

The choice fell on the implementation and utilization of XML. With it's easy handling, platform independence and simple processing capabilities for JavaScript and other languages, XML was the perfect match.

3.2.5 The XML Tutorial Tree

The XML structure for the tutorials is defined as presented in Figure 2. It is divided into three subsections that are presented in detail within this chapter. A tutorial comprises mandatory and optional tag elements. All necessary tags are marked with a leading (1) before their tagname. This is crucial for some elements. For example, although a tutorial with missing <*description*> tag and content will still be loadable into TVB-EduPack, other missing elements like IDs will lead to unpleasant errors, because these IDs are essential for the parsing process when the file is loaded into TVB-EduPack.

First Structure Level – The Tutorial Configuration

The first section, which is illustrated in Figure 3.10, contains the tutorial related tags. This includes <*title*>, <*descriptions*>, possible <*preconditions*> and the arrangement of the tutorial by defining the <*step*> elements. Figure 3.3 already presented the visual result of how the tag elements <*title*>, <*description*> and <*steps*> will get integrated into the GUI. Additionally the authors of a tutorial also have the possibility to set some preconditions for the tutorial. This can be useful if their tutorial is part of a collection or is designed for more advanced users and requires some basic knowledge in specific topics.

It is considered to work with user levels in the future. When a user completes a task, their level or skills could rise, and this level could be used as an indicator for higher or more advanced tutorials. Right now, a <*precondition*> can only be set by directly referencing the title of another accomplished tutorial or demanding a specific page in TVB.

Second and Third Structure Level – The Content Presentation

The second level starts with defining <*step*> elements. Everything between the starting and ending tag of a <*step*> can be seen as objects of the second and third structure level. As the indicator in round brackets shows – a tutorial consists of 1 or more <*step*> elements. The info about a <*label*> and <*step-id*> are required within a <*step*> element. The <*label*> will be the heading and the ID is required for the arrangement and used as a reference for corresponding sub-level elements and animations within the menu.

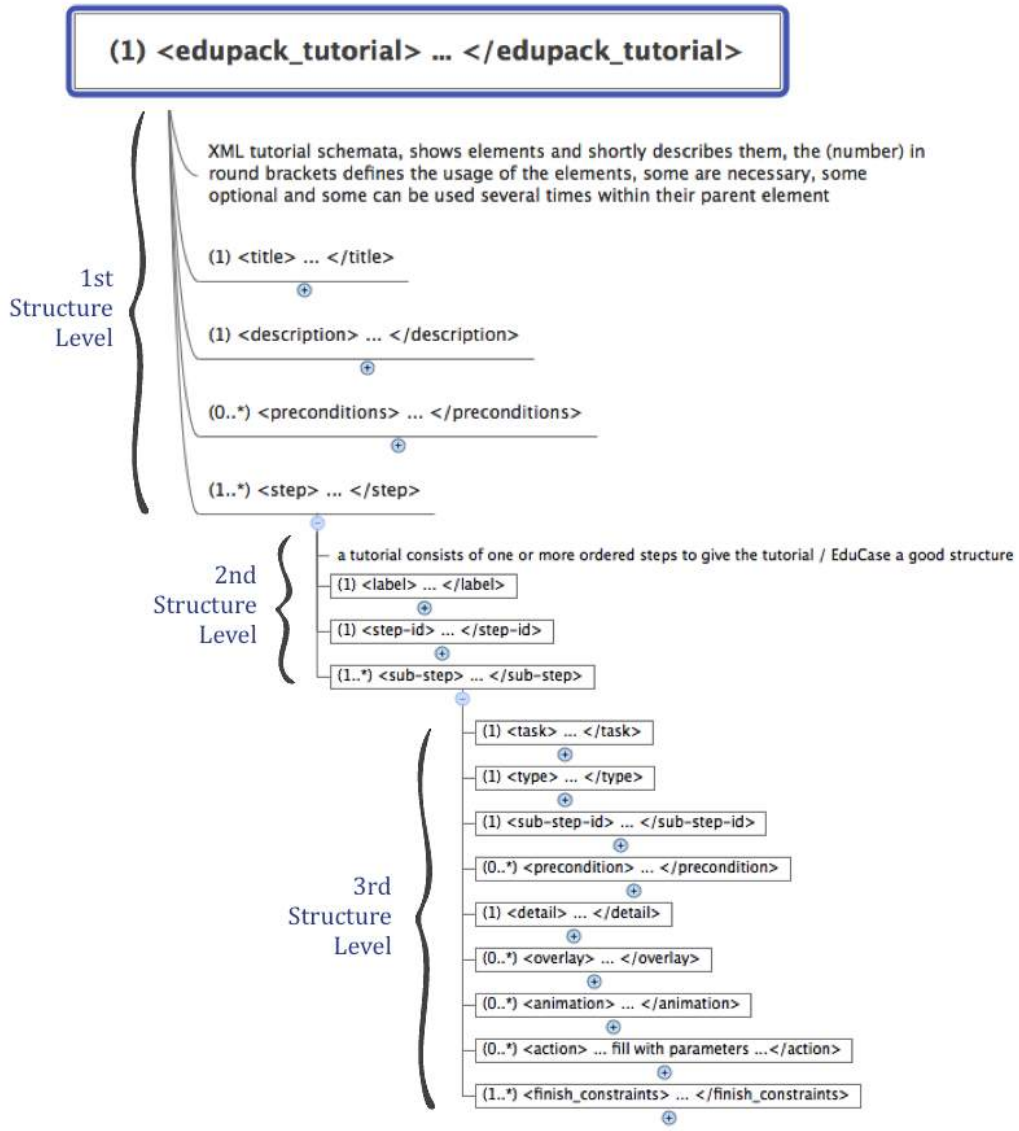

Figure 3.9: Overview of the XML Tutorial structure of all three levels. The following section will describe the structure in more detail.

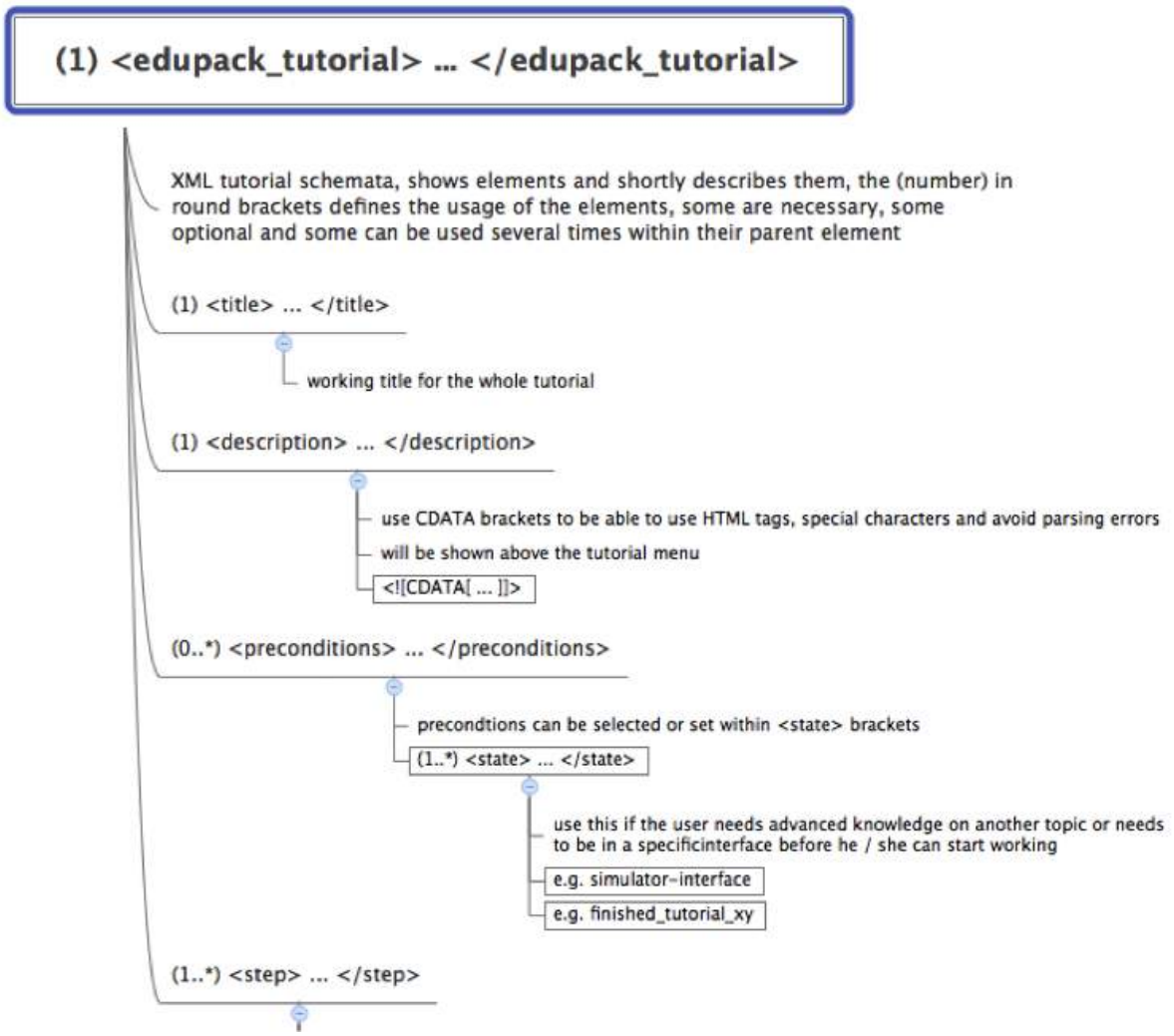

Figure 3.10: First Structure Level <*edupack_tutorial*> of XML Tutorial Tree.

The third level starts with the <*sub-step*> tag. As Figure 3.9 shows, every tutorial has to consist of at least one <*step*> and one <*sub-step*> element. This requirement is set because a tutorial without a task would not make sense. <*Sub-step*> elements also need an ascending <*sub-step-id*>, a title that is written between the <*task*> brackets and a <*type*>. The type is based on the general color-coding scheme that was presented in section 3.2.1 in order to help the user see what the task is about in advance. Additionally, some further elements used in this <*sub-step*> section will also be affected, such as links, overlays and pointer elements. At the moment the description <*type*> is set to green, the parameter <*type*> to orange and miscellaneous to yellow. The standard <*type*> is description.

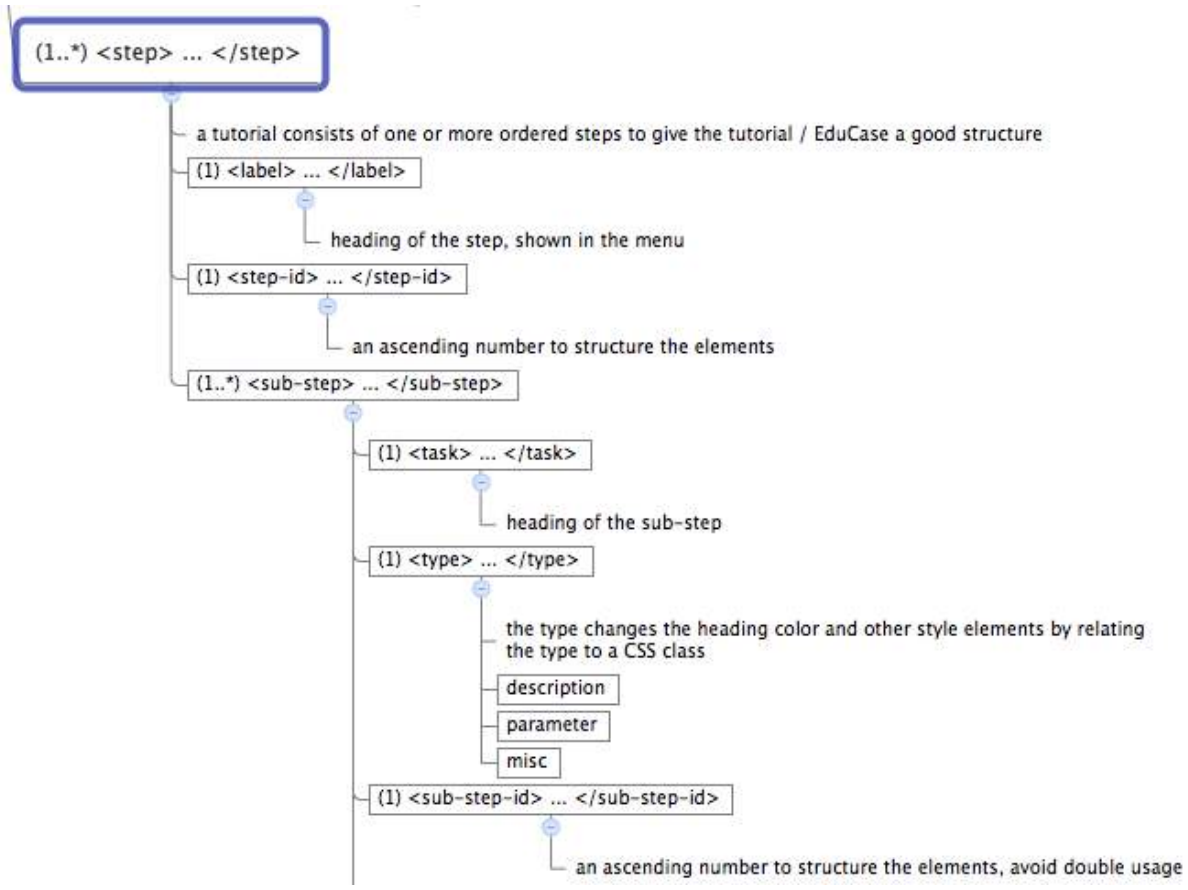

Figure 3.11: Second Structure Level <*step*> of XML Tutorial Tree.

Feature Element - Preconditions

All <*sub-step*> elements will be visible in the GUI when the user of TVB-EduPack has activated their parent step element. When the created <*sub-step*> elements are listed, they are still not active. They will become active when the element is clicked and the content slides down. The following section will now get into the detail about possible integrations and the usage of <*sub-step*> children elements.

When a user activates a <*sub-step*> element, all elements <*preconditions*> will be tested, and the content is only loaded when all of them are fulfilled. Preconditions can be set for each <*sub-step*> individually, see Figure 3.12. This might be helpful when a previous task asks the user to switch to another interface in TVB, so that TVB-EduPack can verify the progress or if the completion of a previous task that is necessary to continue. If a <*precondition*> is not fulfilled, the <*sub-step*> element will be marked as "blocked" and will not open as usual. The author can also write an error message, which would pop up in a small overlay. The error message can be simple text, or if using *CDATA* tags – filled with images as described before.

As soon as all <*preconditions*> are fulfilled, the content of the <*detail*> tag slides down. In here the author has to write the instructions or descriptions for the current task. Figure 3.3 - label F shows how this will look like in TVB-EduPacks GUI.

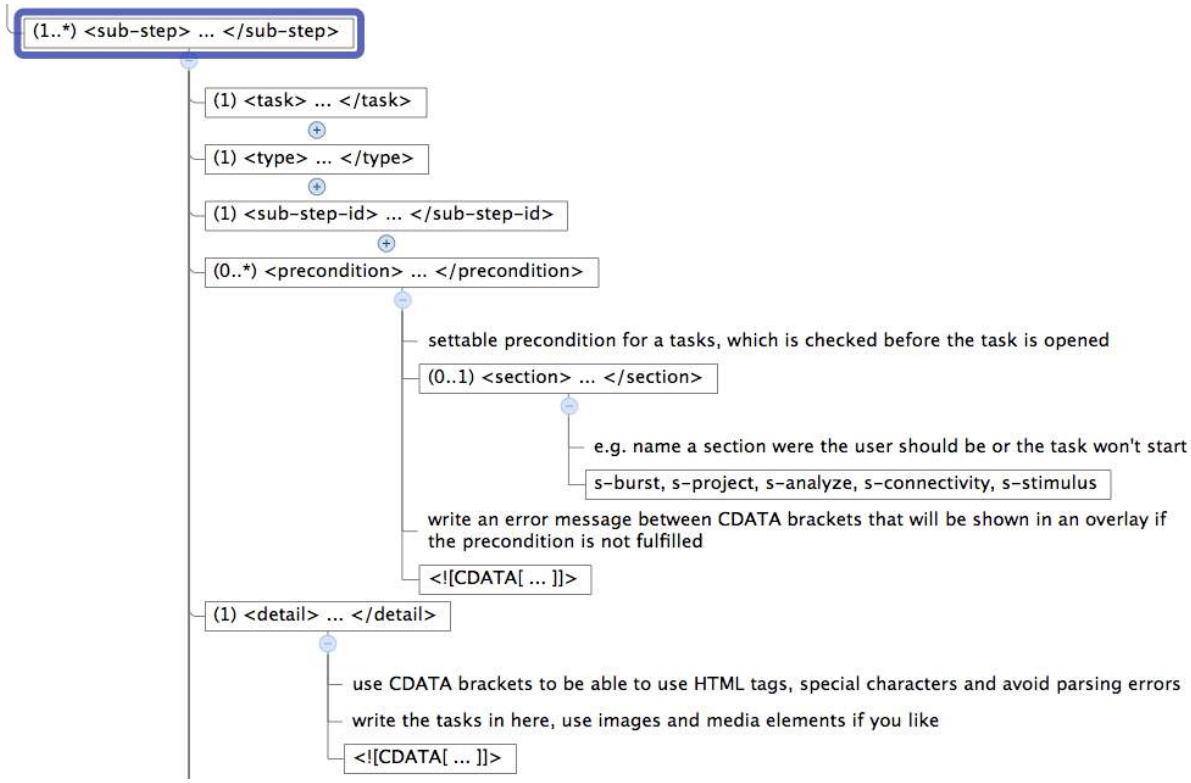

Figure 3.12: Third Structure Level <*sub-step*> of XML Tutorial Tree.

Feature Element - Overlays

Overlays can be really helpful if further material needs to be integrated, an issue needs to be described in more detail or the focus on a particular result needs to be emphasized. Figure 3.13 shows a visual example of the implementation of an overlay object with slider elements. One can still see the TVB GUI in the background with TVB-EduPack and the related overlay link within the task. The integrated links for an overlay will always be marked with a blue "*Info*" button. When the user clicks the link, the overlay pops up, the background fades out and the overlay itself is in the foreground until the user closes it by clicking into the background. The content is written into the overlay's <*content*> tag, and it is possible to work with slider elements as shown in Figure 3.13. One can utilize slider elements to present pre-computed results and therefore emphasize some parameter ranges already explored, see section 4.1 for an application example.

The <*input*> and <*output*> fields within the <*slider*> parent element need to be linked to a corresponding JavaScript method, see Figure 3.14. In the case of the <*output*> field

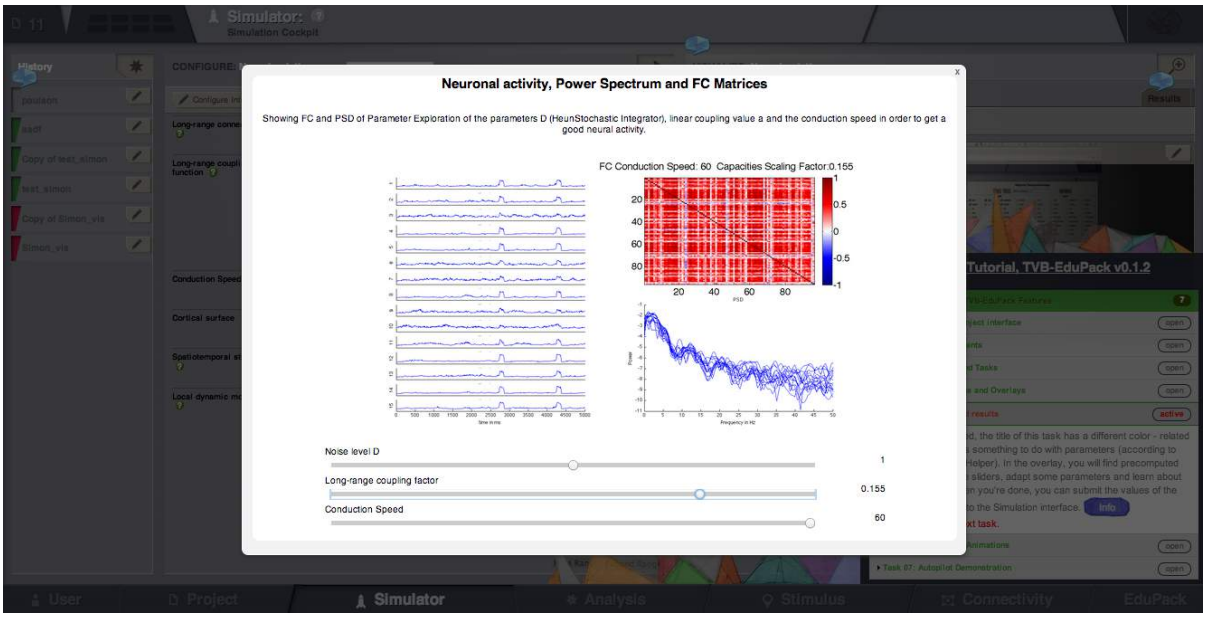

Figure 3.13: Overlay example with pre-computed results and slider elements that facilitate the user to change parameter values and explore the simulation results.

this is not necessary, because this will only show the changing value of the parameter. But the linked method of a <*slider*> element needs to correlate and connect with the proper datasets in the script. Within the examples of pre-computed results, multiple sliders were linked to a set of images that were generated through a parameter exploration process within TVB, see Figure 3.13.

To simplify the integration of slider elements with related data in the future, an editor could be implemented that will automatically generate the code for TVB-EduPack. Then the author would only had to select a set of pictures, define one or multiple slider elements, and the names and the step sizes of the changing parameter values, instead of writing the code by hand.

Feature Element - Constraints

The concept of the <*finish_constraints*> is comparable to the principle of <*preconditions*>. A <*finish_constraint*> can be either set with a manual tag or even get notified automatically by an action primitive that a constraint is fulfilled. If it is set manual, a link will be integrated into the sub-steps <detail> tag, which the user can click to continue. Or the author defines a automatic constraint, which could be related to the task, like waiting for key input or check the users set parameters. It is possible to set one or more <*finish_constraints*>, even combine manual and a defined <*state*>.

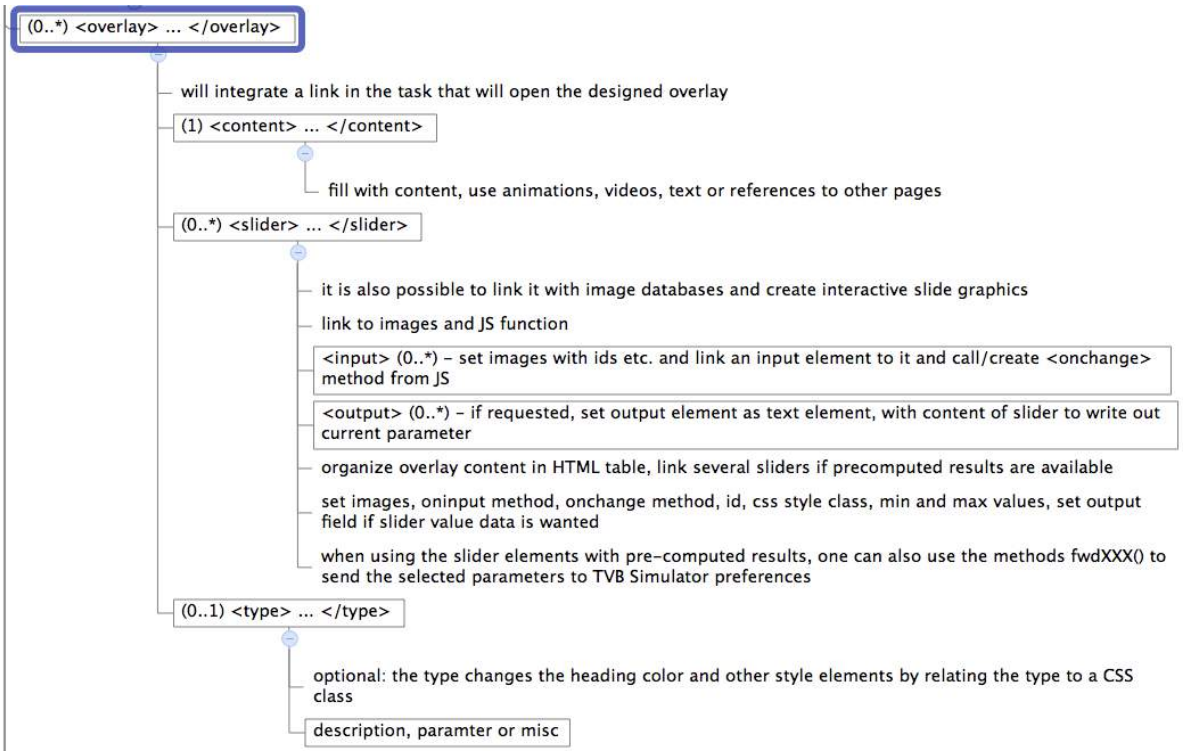

Figure 3.14: Details of an <*overlay*> element.

3.2.6 XML Action Primitives

The developed idea about the action primitives is quite simple:

- reusable
- extendable
- configurable

As with the other XML elements, all elements are reusable and can be integrated into the appropriate level of the XML script. Action primitives will be configurable with parameters or other information, too. But the main idea is that other developers shall also be able to develop new action primitives that match the interests of the developer and teacher.

Without foreclosing the next sections, all presented action primitives are supervised with an action-manager that takes care of all loaded primitives for the current task. This manager gets informed as soon as a new task is opened, or an old one is closed. This is important because the manager has to delete task related primitives from the system and GUI. Furthermore, it takes care of all active primitives, checks their states and reacts with the corresponding action.

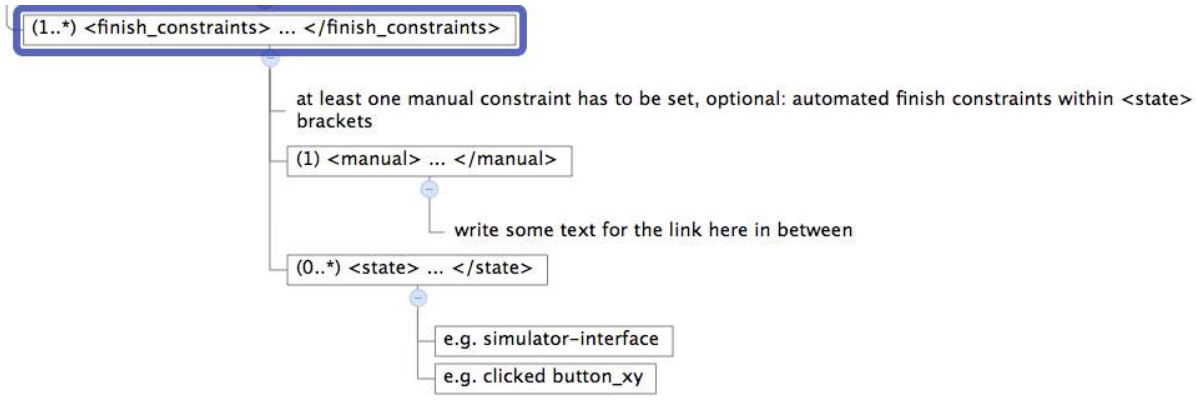

Figure 3.15: Details of the <*finish constraints*> object.

Action Elements and the Concept of Primitives

The event-oriented modular concept that goes in hand with the general idea of two-way-interaction comes to life with the action primitives. With the generalization of potential interactive elements into action primitives, one gets a system that is flexible, event oriented (either to the task or the user), and allows the combination of several primitives in the end.

The idea is that all primitives will be related to a specific level in the XML structure. When one of these structure levels activates, the action primitives will be loaded into TVB and stay active until they are fulfilled or the structure element is closed again.

The management of active and inactive structure elements, and the already loaded primitives, are managed with JavaScript. Whereas some primitives are weak – other primitives are strong and need to be completed in order to continue with the tutorial. This is all organized in an administration script that allows to add, delete, change or set status of action primitives and manage them over time within a tutorial, see section 3.2.7.

3.2.7 Scope of Action Primitives

The action primitives are divided into different scopes. These scopes only group and structure the elements to emphasize their function. The biggest scope of action primitives is the listed groups in the Application Control. Due to the functionality of TVB-EduPack, these are one of the most important elements. But some of the Application Control Action primitives that read, write, highlight or compare datasets are also bound to the primitives that work together with User Interaction or Flow Control primitives – which results in the two-way-interaction model, see section 2.2.

The most important thing about the primitives is that they work independently. They can be integrated and loaded within the XML structure, but they act as single elements. One can load and control one or more primitives within a single task, wait for them to

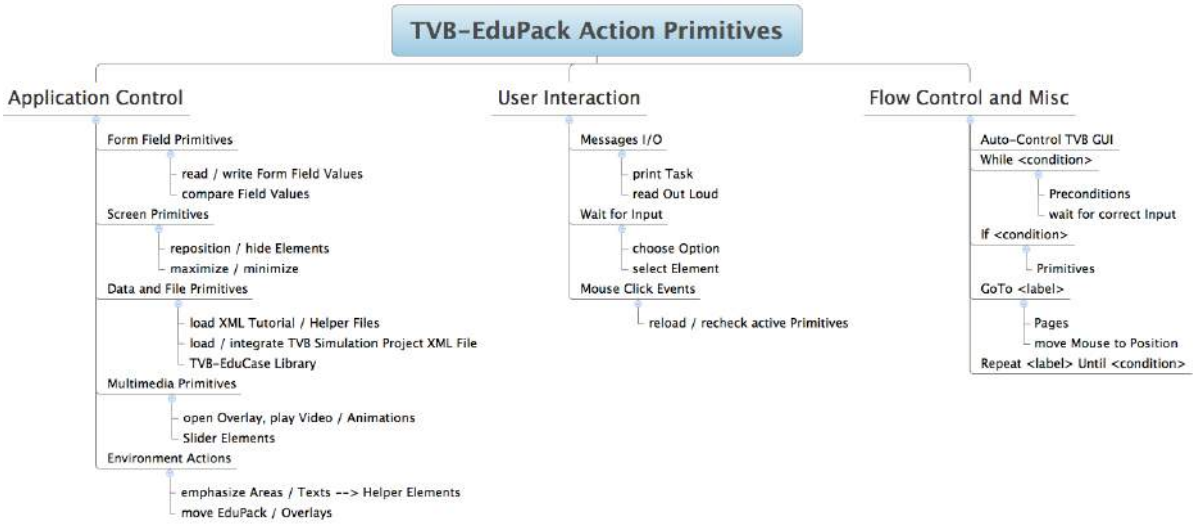

Figure 3.16: General overview of the scope of Action Primitives.

be fulfilled, combine them or use them to emphasize something on the screen. Singular primitives does not mean that they are not working together, only that they can be used alone, but some of them, like flow control structures, are related to other processes and checking other primitive statuses or waiting for inputs at form fields. Most of the listed elements in Figure 3.16 are already integrated into the first prototype of TVB-EduPack and can be found in the implemented example tutorial in the section 4.1.

3.2.8 Three Action Primitives in Detail

I will now present three examples for action primitives in this section. The whole description, parameter and attribute settings can be found in the appendix, section 6.1.

Action Primitive: task_overlay

The <*task_overlay*> primitive is mainly used as part of guided tutorials. One can use it to point to a selected object on screen. TVB-EduPack will then add a type-colored TVB-Helper element with an automatically added task id number at the selected position. Equivalent to TVB-Helper elements, the author of the tutorial can select a type for the element and fill the primitive with content that will be provided in an overlay. For emphasizing an element on screen, the same information as for TVB-Helper element are necessary – *section*, *group* and *variable_name* of the object.

One can integrate more than one of the <*task_overlay*> elements within one <sub-step> task, for example to describe some TVB functionality or as done in the example study: to emphasize the buttons and elements that need user interaction or input within the tutorial, see section 4.1.2. Another possible application would be to present results

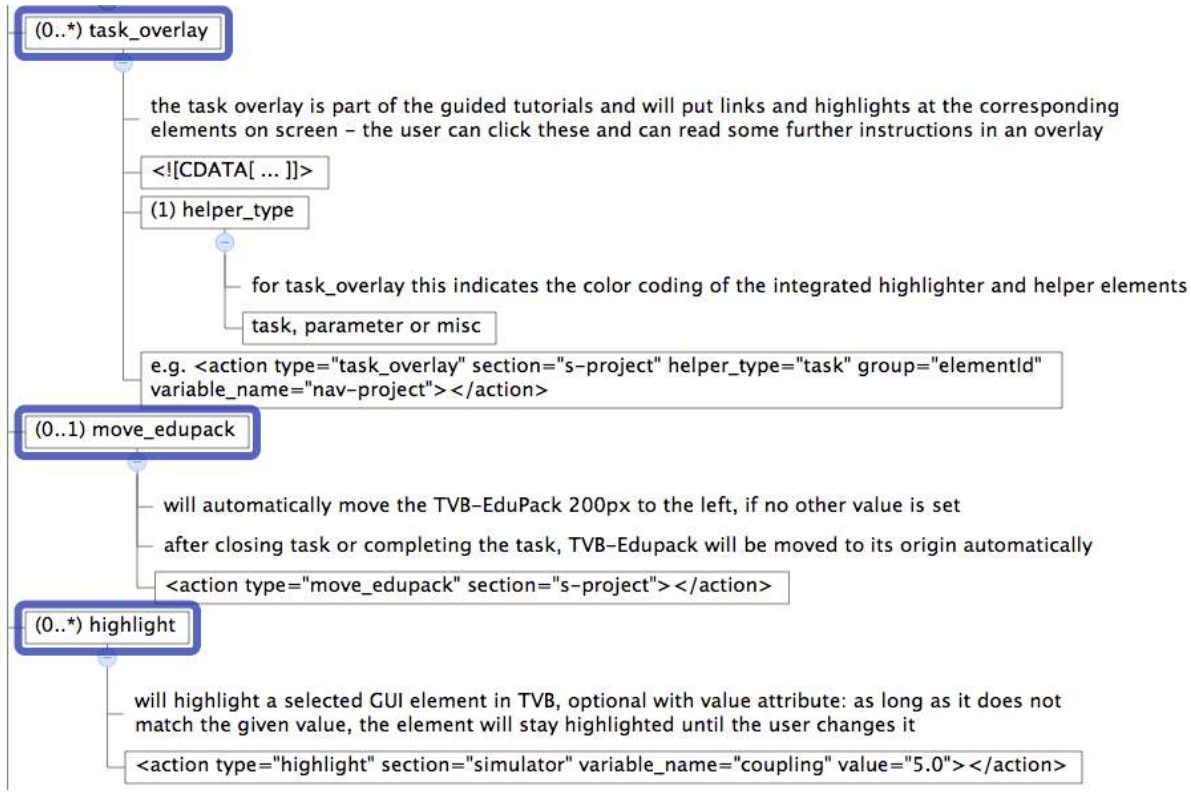

Figure 3.17: Detail of three selected action primitives task_overlay, move_edupack and highlight.

within a simulator tutorial for specific parameter ranges, because these <*task_overlay*> primitives can be filled exactly like the <*overlay*> elements. They can also be seen as additional information to the task that is written in the TVB-EduPack menu – only that they are directly linked to the selected object on screen and the user has to open the <*task_overlay*> element to read the description.

Action Primitive: move_edupack

The <*move_edupack*> primitive is useful to use when you know that TVB-EduPack menu overlay is hiding some elements in the background. For example when the user has the task to operate in the lower right screen corner, the elements are most likely hidden behind TVB-EduPack. This action primitive allows the author to move TVB-EduPack to another corner or position. In contrast to other action primitives that can be integrated multiple times into one <*sub-step*> level, the <*move_edupack*> will only work once in a <*sub-step*> element. After the task is completed, TVB-EduPack will automatically move back to its origin position unless the next task includes a <*move_edupack*> primitive as well. This action element is most likely used in combination with other action primitives, because its movement allows the user to click elements behind TVB-EduPack without closing it. The latter would also be a possible method to solve the task. Figure 3.17 shows

the attribute *moveX*, which is set to a default value, that means that TVB-EduPack will move 120 pixels to the left. Other values are possible and refer to pixel values.

Action Primitive: highlight
This was one of the first action primitives developed. With the <*highlight*> functionality, one can emphasize elements that have wrong parameters or need some attention during a tutorial. But these primitives have their disadvantages when highlighting a button, because the highlight function only creates a rectangular element that lays over the selected element – and this does not look good on integrated buttons in TVB. It is more suitable on text or form field elements.

The complete attribute list for all implemented action primitives of the first TVB-EduPack prototype can be found in the appendix, chapter 6.1.

3.3 Package 3 - Implementation of TVB-EduPack

While starting TVB with an integrated TVB-EduPack version, a basic XML sheet with the menu and design reference is loaded in the background. When the user starts TVB-EduPack from the lower right corner, the related files that were already parsed, and TVB-EduPack, become visible instantaneously. When the user has worked with TVB-EduPack before, the stored settings are also loaded and integrated so that they can continue with the last saved configuration and progress.

3.3.1 Architecture of TVB-EduPack

As shown in Figure 1.4, the Application Interface with the web based GUI is directly linked to the Web Server Controller, which is then connected to the Flow Managers in the Backend of the system. The Flow Manager works and coordinates all other system components such as the Uploader, Simulator, Analyzers, Visualizers or Data Types of TVB. Most important elements for the integration of TVB-EduPack are the generated Web Interface structures, the interface for user input, and the Web Server Controller itself.

TVB-EduPack integrates into the Web Server module of TVB, because this is the interface to the graphical user interface. From there, TVB templates were adapted so that the scripts and TVB-EduPack related templates became integrated. Figure 6.2 shows adapted modules and selected file examples of TVB in a mixture of blue and orange. Grey modules are untouched TVB modules and blue elements represent files and packages of TVB-EduPack. The diagram shall give an overview of modules already discussed, like the design classes or XML structure and how independent they are from the system in general. The workflow of TVB is only interrupted when e.g. action primitives check

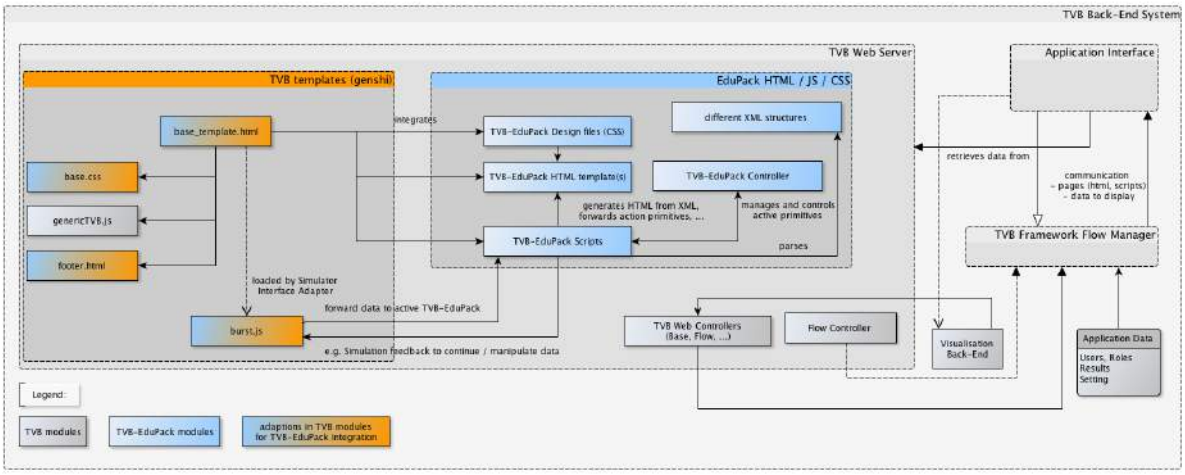

Figure 3.18: Section of the integrated TVB-EduPack modules into the architecture of TVB (the diagram only shows an extract of relevant modules from TVB).

for a parameter configuration for a re-simulation of given results and the user wants to start the simulation or analyzer with wrong set parameters.

To integrate TVB-EduPack into TVB the *base.css* Design file was adapted with some design specifications and information for the link in the footer menu. The *footer.html* template itself also needed an adaption of one line of code. To facilitate action primitives from TVB-EduPack Control Manager to interrupt the workflow in TVB at interesting points, check methods were integrated within the main scripts, e.g. for the simulator. These methods send signals to TVB-EduPack in order to check if there are correlated active action primitives. If this is the case, the workflow is either interrupted with a corresponding message for the user or continues as usual. When TVB-EduPack is not active, TVB works as normal.

3.3.2 XML Parser

According to the different presented XML structures, an HTML template for TVB-EduPack needs to be generated and integrated into the *base_template* of TVB. This is the task of the parsing scripts. At the moment they only work with the presented structure and do not check for errors. If the structure contains errors, this may lead to errors that cannot directly be seen. As a result, they may not affect TVB, but the functionality of TVB-EduPack. The parser starts to work with either the last loaded XML tutorial or the start menu when nothing is saved in the *localStorage* of the browser. Additional state variables save the current open and finished tasks. These values may be checked by some task-related preconditions and are also saved for the *localStorage*

configuration so that the user can continue whenever he stops working with TVB and TVB-EduPack.

While the menu with all the content is loaded and generated for TVB-EduPack, the action primitives and the content of the <*overlays*> are only loaded when the user activates the elements.

Parsing Techniques

With the utilization of different parsers it is easier to extend the XML structure. Therefore a parser was written for each of the two XML structures - for TVB-EduPack Helper elements and the XML tutorials. Two more parsers for the task-related preconditions and action primitives were necessary. While the parser for the Helper elements and TVB-EduPack tutorials creates HTML DOM Tree elements and integrates them into TVB-EduPack once, the parser for the task related methods is event oriented and checks for preconditions plus action primitives whenever a task is opened. For all found preconditions and action primitives the attributes are saved or forwarded to continue with the processing. When one or more preconditions are not fulfilled, this will be shown with a default, or if defined - a specialized message for the user and all other checks are cancelled.

After extracting the action primitives, they are forwarded to TVB-EduPack Controller. The controller checks for similar active primitives and either ignores the new primitive, adapts its lifecycle, or adds it to the list of current active action primitives. This was problematic in the beginning, because when there was no checkup for other active primitives, and every action primitive was used as single object, many primitives of the same type could be added by opening and closing the same task over and over again. This resulted into many highlighting elements at the same position or to a TVB-EduPack that moved out of the screen. It is now a task of the controller to manage the active primitives when the user starts or finishes a task. But it also needs to review them when the user updates form fields or clicks in specific areas. Some of the users actions can trigger TVB-EduPack Controller to activate and review active instances or even interfere in TVB workflow when it is necessary, see Two-Way-Interaction in section 2.2. A quite similar process is started when the user closes a task or e.g. sets the correct parameter – then the TVB-EduPack controller is called and removes the related action element.

3.3.3 Debugging of TVB-EduPack

As already mentioned before, TVB-EduPack does not check for errors or misconceptions of the imported and integrated XML schemes. As author of a tutorial one has to make sure that first - the structure and utilized attributes match with the tags as presented in this thesis, second – that the utilized elements are used in the correct structure level and third – that all tags have an enclosing tag with leading slash:
<*tag attribute1="default"*> content </*tag*>.

Especially for the third point, tools are available on the internet that validate the XML structure. In this case I can recommend the online tool http://xmlgrid.net/, which also lists some debug information if the file is not well formed. It also has a nice representation of the different structure levels and details. As a matter of fact, developers and authors of TVB-EduPack tutorials or extensions will be aware of the software development life cycle[15], but up until now - the testing needs to be done manually.

A debugging functionality within TVB-EduPack is a feature that could be considered in a future release.

[15]Software Development Life Cycle:
à requirements engineering à design à development à testing à maintainance à

4 Practical-Case and User-Evaluation

"I have always wished for my computer to be as easy to use as my telephone; my wish has come true because I can no longer figure out how to use my telephone."
Bjarne Stroustrup

In this section I will present a demonstration for a TVB-EduStart and TVB-EduCase scenario. I will start with describing two different scenarios in detail with focus on selected XML elements and action primitives. In order to conclude this thesis I will present a user evaluation with correlated feedback for the first prototype of TVB-EduPack.

4.1 Practical-Case

The following example shows selected code snippets and screenshots taken from TVB-EduPack. It will start with a TVB-EduCase, an easy to follow step-by-step guide that asks the user to create a new project within TVB. The second part is also a guided step-by-step to start a simulation, but it is not so straightforward. It offers the user more freedom of choice and already tries to motivate the user with additional material. Therefore it makes use of visual explanatory animations and pre-computed simulation results.

Both examples are written into one file and are typical use case scenarios for TVB. Since the first tutorial is really direct and does not give much room for explorations, the second one tries to teach and show the user something new so that they can immediately start testing it out in TVB.

The complete source code is attached to the appendix, section 6.2.

4.1.1 Part 1: TVB-EduStart Example

Listing 4.1: The source code excerpt shows the tutorial related configuration and definition of the first <step> and <sub-step> element with integrated task and action primitive.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <edupack_tutorial>
3   <!-- title for this tutorial and tutorial_name for internal reasons e.g. preconditions -->
```

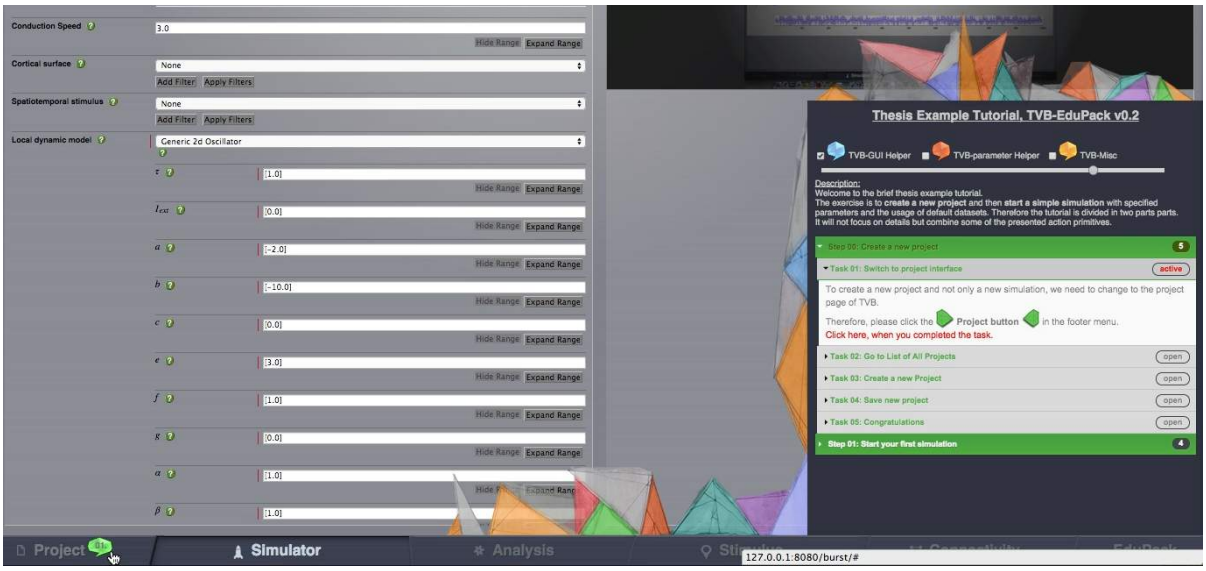

Figure 4.1: Exemplary screenshot of TVB Web-Interface that shows the first task of TVB-EduStart Tutorial.

```
<title>Thesis Example Tutorial, TVB-EduPack v0.2</title>
<tutorial_name>tutorial_thesis</tutorial_name>

<!-- preconditions for whole tutorial, here deactivated -->
<preconditions>
  <!-- <state>finished tutorial_import</state> -->
</preconditions>

<!-- description text above menu -->
<description>
  <![CDATA[
    Welcome to the brief thesis example tutorial.<br>
    The exercise is to <b>create a new project</b> and then <b>start a simple simulation</b> with
    specified parameters and the usage of default datasets. Therefore the tutorial is divided in
    two parts parts. <br>
    It will not focus on details but combine some of the presented action primitives.
  ]]>
</description>

<!-- here starts the first step with all its sublevel structure elements -->
<step>
  <label>Create a new project</label>
  <step-id>00</step-id>

  <!-- here starts the first sub-step, no preconditions but using task_overlay primitive
    to emphasize the task - here to switch into the Project page of TVB -->
  <sub-step>
    <task>Switch to project interface</task>

    <!-- type is "description", later in the tutorial it will change to "parameter" -->
    <type>description</type>

    <!-- sub-step id -->
    <value>01</value>

    <!-- description of the task -->
```

```
40      <detail>
41        <![CDATA[
42          To create a new project and not only a new simulation, we need to
43          change to the project page of TVB.<br/>
44          Therefore, please click the
45          <img src=/static/js/icons_green11.png width=5% valign=bottom>
46          <b>Project button</b>
47          <img src=/static/js/icons_green11_2.png width=5% valign=bottom>
48          in the footer menu.
49        ]]>
50      </detail>
51
52      <!-- action primitive "task_overlay", showing a numbered Helper element
53        (number of the current sub-step) and an additional message presented in an overlay.-->
54      <action type="task_overlay" section="s-project" helper_type="task"
55              group="elementId" variable_name="nav-project" read="true">
56
57        <![CDATA[
58          This is the first task you have to solve, <br>
59          click the <b>Project</b> button to switch to the Project page.<br><br>
60          <img src=/static/js/tutorial_preview_project.png width=400>
61        ]]>
62      </action>
63
64      <!-- no auto completion within this tutorial, if the user clicks the link, the next substep
65        will be opened and the previous task will be marked as finished -->
66      <finish_constraints>
67        <manual>Click here, when you completed the task.</manual>
68        <!-- <state>none</state> -->
69      </finish_constraints>
70    </sub-step>
```

When the user starts TVB-EduPack, the first task of the tutorial is presented. They will see the menu structure, as in Figure 4.1. The documented implementation of this part can be found in Listing 4.1.

The <*edupack_tutorial*> is defined without preconditions. All parameters and elements are set as described in section 3.2.5. The <*sub-step*> is defined as description type, uses images in the detail field for the menu and integrates helpful text in the action primitive <*task_overlay*>. The whole example tutorial does not make use of automated <*finish_constraints*>, so all <*sub-step*> elements will need a manual link as shown in Listing 4.1.

The following integrated <*step*> and <*sub-step*> elements work with an equal setup as in Listing 4.1 and only further additions or related changes will be listed in the next examples.

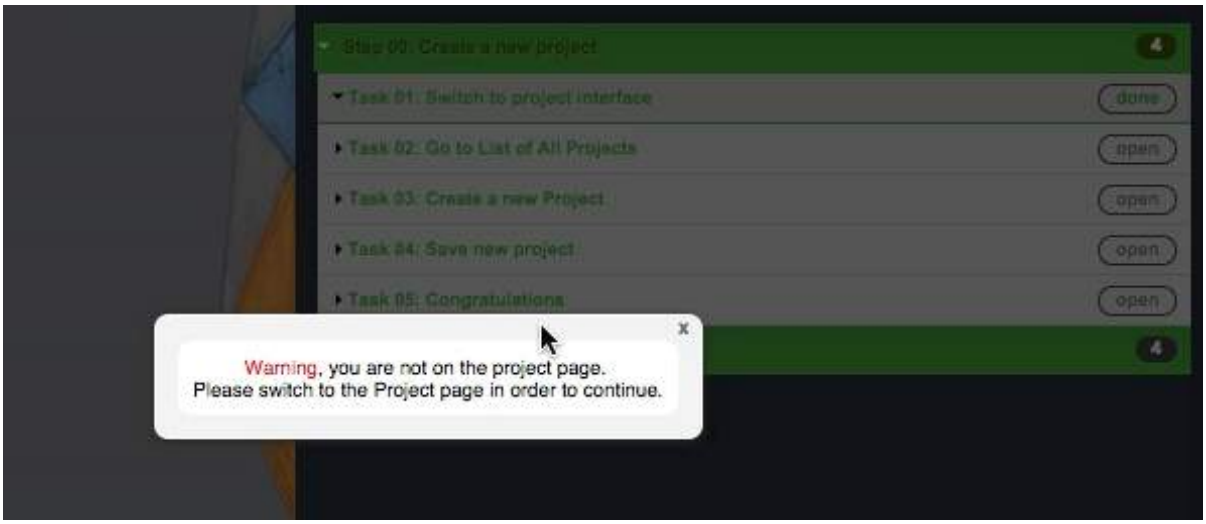

Figure 4.2: The User will get an Error Message when a task-related <*precondition*> is not fulfilled.

Listing 4.2: Using a <*precondition*> makes sure that the user is in the correct menu and able to continue.

```
<sub-step>
  <task>Go to List of All Projects</task>
  <type>description</type>
  <value>02</value>

  <!-- related to the previous task, the user should now be on the project page,
    otherwise he / she will get a warning message and is not able to open the sub-step -->
  <precondition section="s-project">
    <![CDATA[
    <font color=red>Warning</font>, you are not on the project page. <br>
    Please switch to the Project page in order to continue.
    ]]>
  </precondition>
```

The first task asks the user to switch to the project page of TVB. When the user does not follow the instructions and continues nevertheless, TVB-EduPack will open a warning message, see Figure 4.2. This warning is related to the defined precondition in the second <*sub-step*> element, see Listing 4.2.

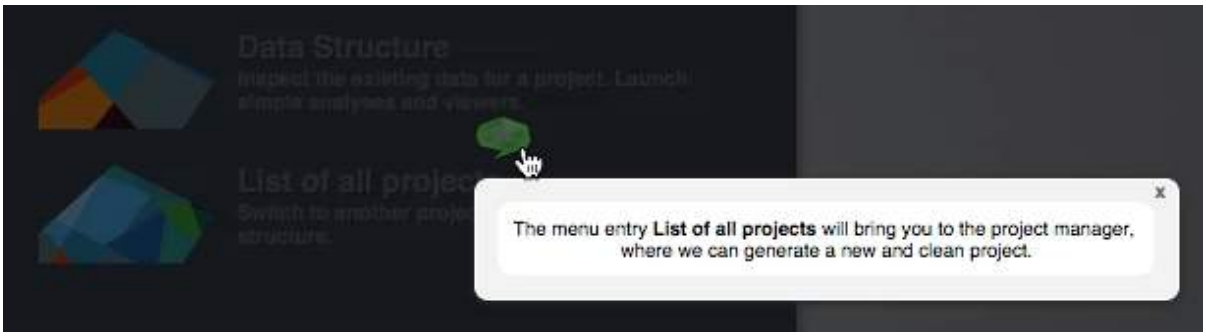

Figure 4.3: The <*task_overlay*> Element of the second task indicates the Menu Button the User has to click.

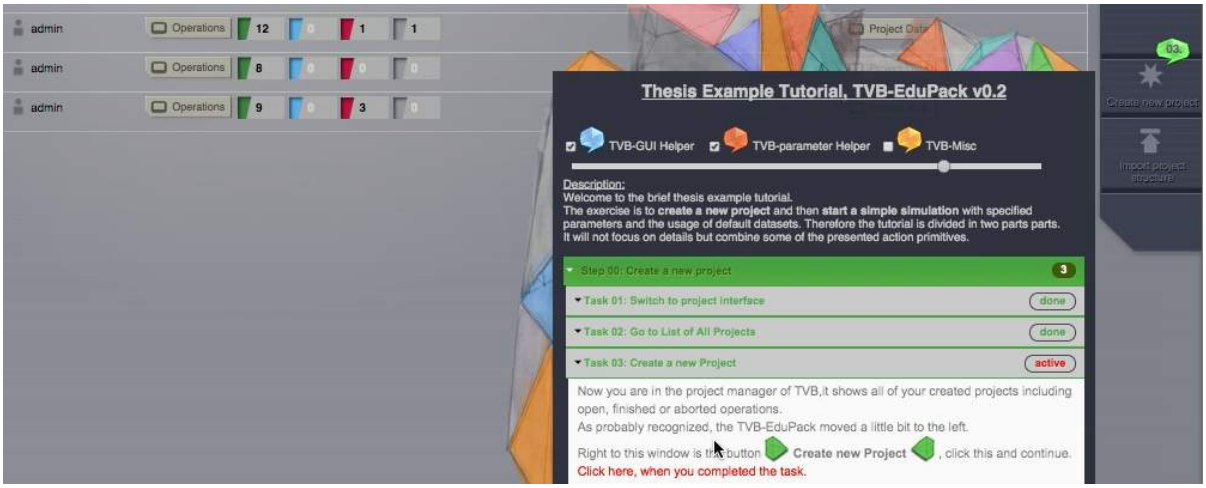

Figure 4.4: TVB-EduPack is able to move and expose the hidden Buttons if the primitive <*move_edupack*> is used.

Listing 4.3: Action primitive <*move_edupack*> will move the menu of TVB-EduPack 120 pixels to the left.

```
1  <!-- action primitive to move TVB-EduPack some pixels to the left in order to uncover hidden
       elements -->
2  <action type="move_edupack" section="s-project">
3  </action>
```

The third and fourth task use the action primitive <*move_edupack*> with the default configuration to move TVB-EduPack, see Figure 4.4 and the command in Listing 4.3. Both times the TVB-EduPack menu hides a button that is necessary for the related user task.

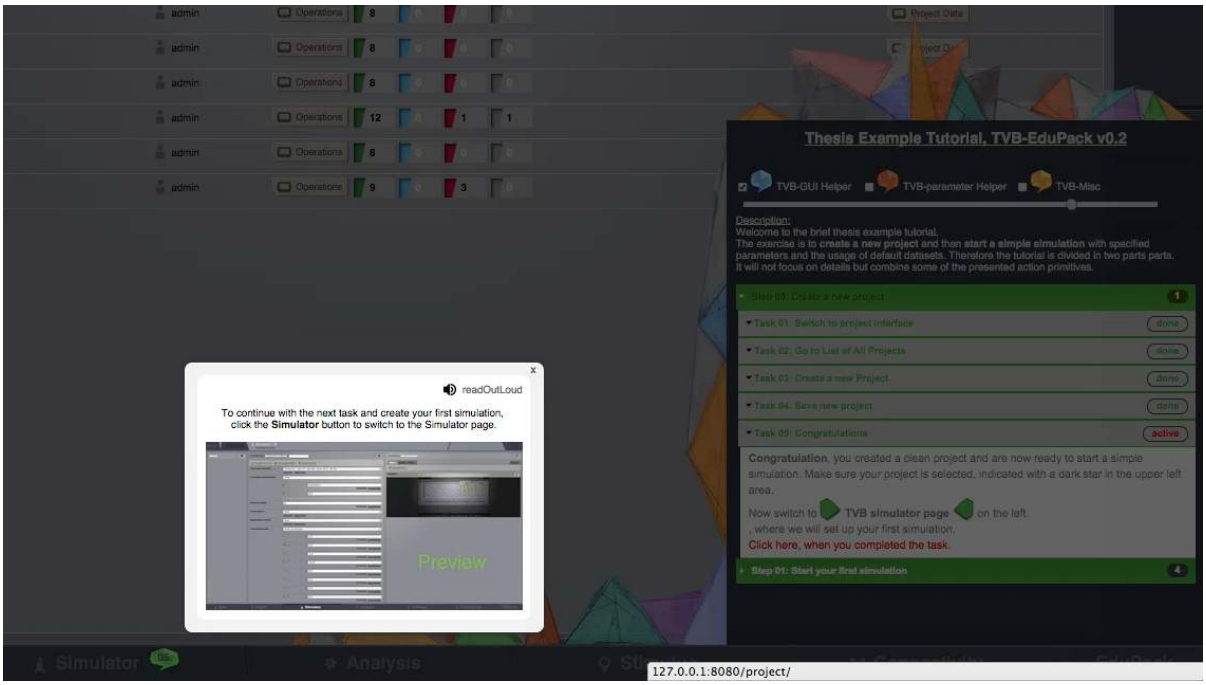

Figure 4.5: Only one Task left for the First Part of the Tutorial. The Screenshot shows the open *<task_overlay>* at the related Position.

Figure 4.5 shows that the user has almost finished all tasks of TVB-EduStart and can now continue with TVB-EduCase.

4.1.2 Part 2: TVB-EduCase Example

In this tutorial the user will learn how to start a new simulation with a specific parameter setting. The values are based on a previous evaluated parameter range exploration, and can be previewed in an overlay with two slider constructions. This will allow the user to compare results of different parameter settings before they run the simulation by themselves.

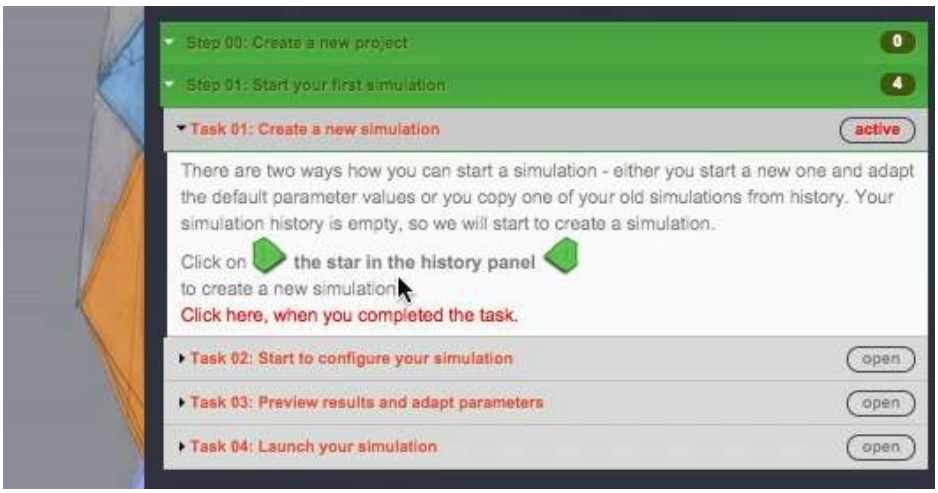

Figure 4.6: Description of TVB-EduPack menu: After a User finished all tasks of a section, the number should be zero. Here, the User opened the next section - it indicates four open task while one of them is already active. All elements are colored orange as indicator for the parameter-type related tasks.

Listing 4.4: The *<task_overlay>* action element is extended by a read-out function.

```
<!-- burst-history link -->
<action type="task_overlay" section="s-burst" helper_type="task"
       group="elementId" variable_name="input-burst-name-id" read="true">

  <![CDATA[
    Give your simulation a significant name and learn about
    TVB-EduPack Helper elements by activating the help for the parameters.<br><br>
    <img src=/static/js/tutorial_preview_helper.png width=400>
  ]]>
</action>
```

Within the last tasks, I integrated the feature *readOutLoud* in the upper right corner of the *<task_overlay>* element, see Figure 4.7 and the related code in Listing 4.4. This can be used by integrating the attribute *read="true"* to the action primitive.

Listing 4.5: The *<highlight>* primitive can be used either to highlight a parameter until it matches a defined value (here: conduction_speed) or to highlight an object throughout the time the task is active (here: coupling parameter), visible in the background of Figure 4.8.

```
<!-- example for highlight primitive, with defined and undefined value-to-be -->
<action type="highlight" section="simulator" variable_name="conduction_speed" value="40.0"></action>
<action type="highlight" section="simulator" variable_name="coupling_parameters_option_Linear_b"></
    action>
```

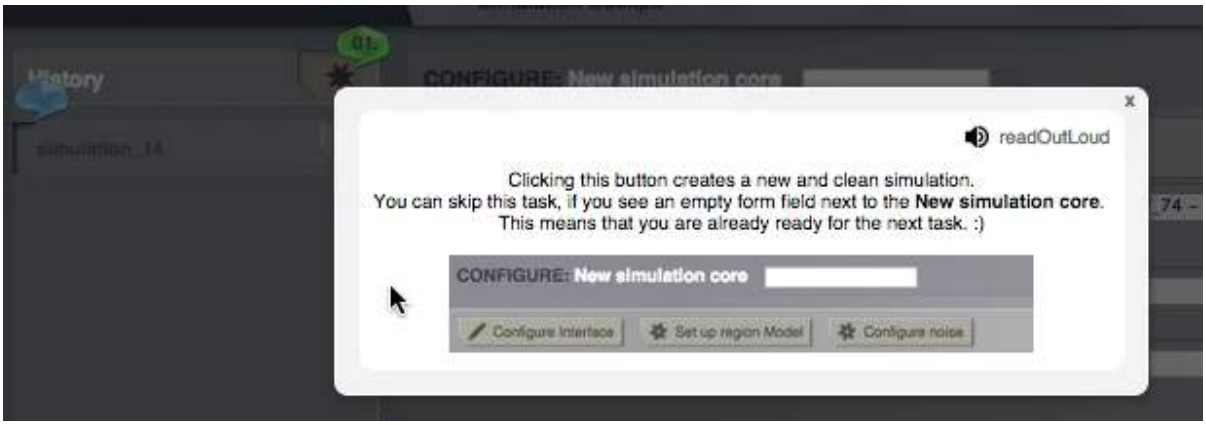

Figure 4.7: The Tutorial shows the User how to create a new Simulation and offers to read the text out loud.

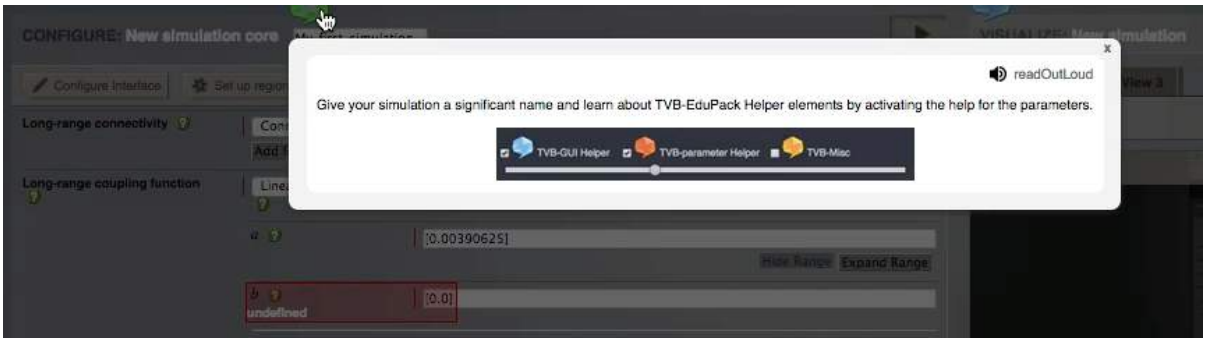

Figure 4.8: The User gets informed about TVB-EduPack Helper Elements and a Task related <*highlight*> Primitive is visible in the Background.

After creating a new simulation, TVB-EduPack informs the user about the additional Helper elements that describe parameters and other GUI elements in TVB.

The feature of pre-computed results can be used for beginners so that they can get started. But is even more valuable for advanced users when they can take advantage of other groups' research findings. The groups could create and offer their results in the proposed TVB-EduCase library. TVB would be an ideal platform to share experiences about different computational models and data types. In this tutorial it is only used to motivate the user and save computational power for simulations that lead to good results.

The presented TVB-EduCase concludes with the suggestion to start a simulation with the selected parameters. The user could now continue working with the next tutorial, such as analyzing results.

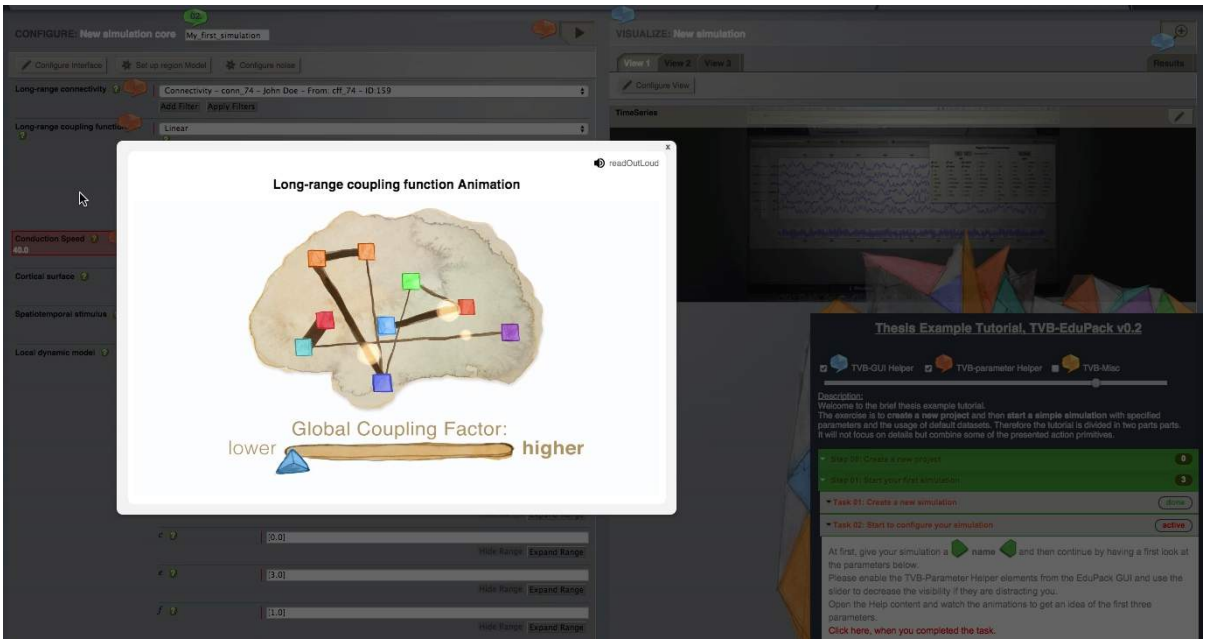

Figure 4.9: Integrated Helper Videos can improve the Motivation and Learning Experience for the User.

All presented figures are taken from the first prototype of TVB-EduPack. It was my intention to give an impression of two scenarios that visualize the use of the implemented features. The next section will continue with a brief user-study that will use the implemented tutorials.

4.2 User-Study Evaluation

As a result of engineering requirements, software design decisions and the implementation of a flexible, XML-based scripting toolkit – the first prototype of TVB-EduPack was presented in this thesis. To conclude and review the practical outcome, I asked seven people to evaluate an example application of TVB-EduPack.

I selected a group of seven people for testing the application. Three of them had a background in computer science. All of them were asked to work with both TVB-EduStart and TVB-EduCase tutorials. The developers were also asked to use a prepared TVB-EduPack XML tutorial script and enhance it with some features.

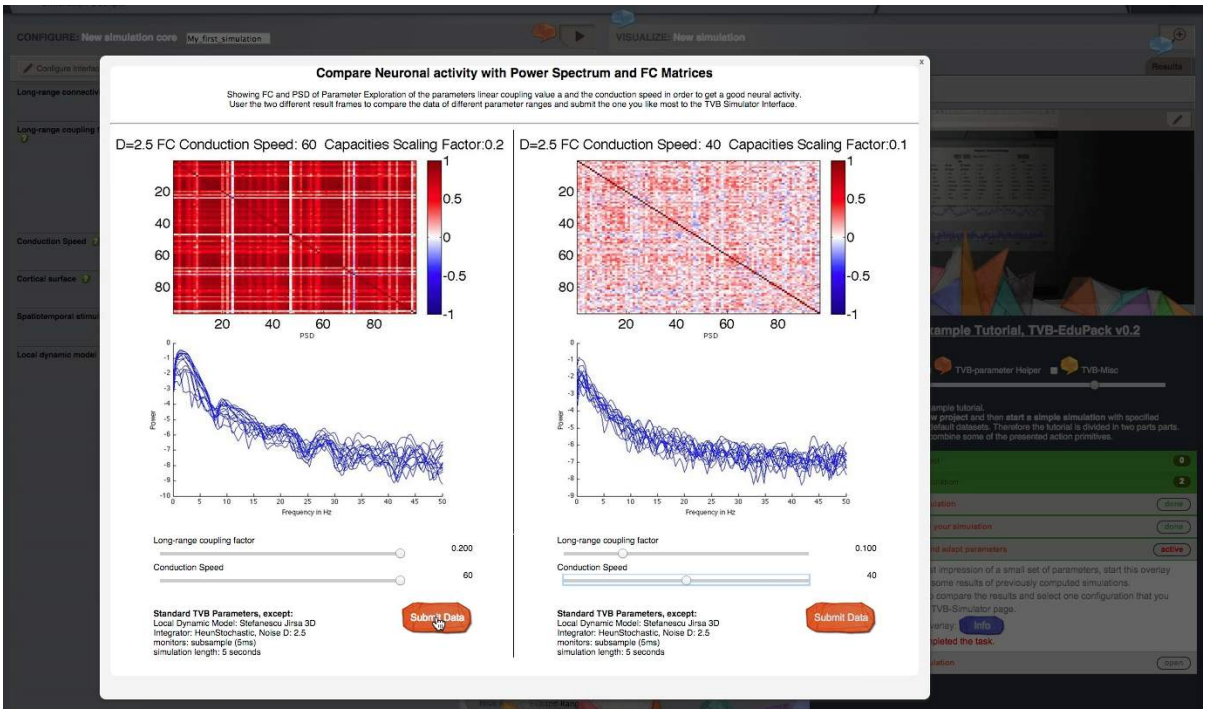

Figure 4.10: To simplify and help new users getting started with TVB, they can work with pre-computed simulation results. In the tutorial they are presented in an overlay and can be submitted to the simulator of TVB. The code for the overlay can be found in the appendix, section 6.2.

4.2.1 Task 1 – TVB-EduStart Scenario

The idea was to get feedback and to watch people get started working with a plain version of TVB, so TVB-EduPack was not integrated. All seven persons had no experience in working with TVB. After introducing them to TVB software, its functionality and features, I asked them to create an empty project. The given task is similar to the prepared TVB-EduStart tutorial from the previous section. All users were starting from the simulator page and did not have access to the written TVB-User manual.

I defined a generous time limit of five minutes for the task and everyone achieved it in time. Only one of the users found the direct way instantaneously, as it is presented in the tutorial. But everyone found the correct links after exploring the graphical user interface. When they finished and changed back to the simulator page, five out of seven were unsure if their created project was the active selected project. This and similar problems with lack of knowledge may lead to more problems in the future. If the users had known that a marked star indicates the active project, they would have seen that their created project was automatically selected.

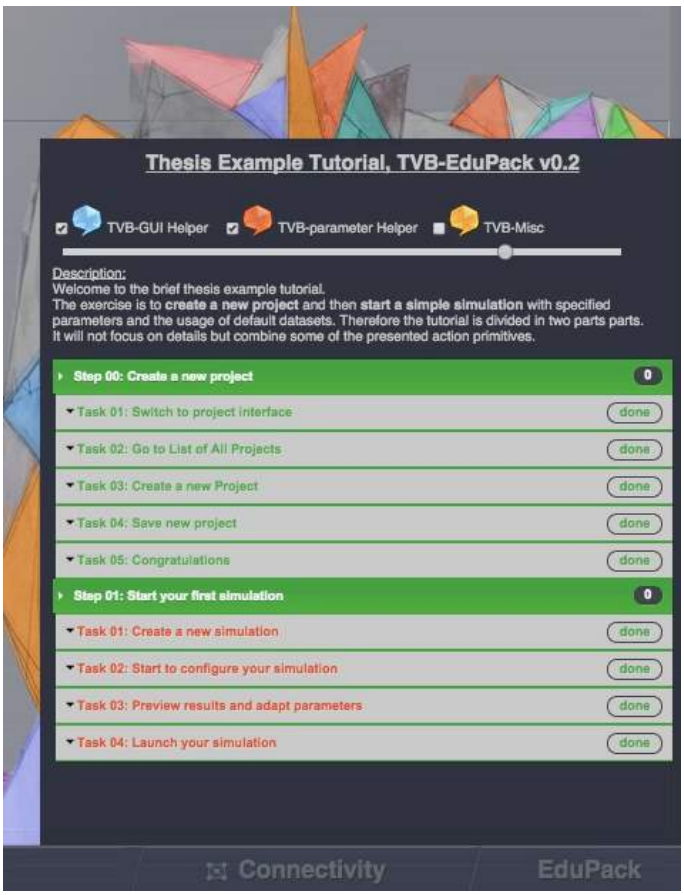

Figure 4.11: Now that the Tutorial is finished, the progress is saved in TVB-EduPack and the User can continue working in TVB - with or without using TVB-EduPack.

Task 1 with TVB-EduPack

Subsequently the second task was to do the same task again, this time with using TVB-EduPack. Right from the beginning all persons felt more confident. With more detailed instructions, most of them worked faster without loosing track. Two of them had problems recognizing the green elements as guides for the tutorial in the beginning, and three were irritated by the not automatically continuing tasks of TVB-EduPack. Obviously everyone succeeded in the task again, but this time they also learned something about featured GUI elements of TVB.

I did the study on a 13-inch laptop with a resolution of 1440 x 900[16] and some users suggested to increase the size of icons and fonts of TVB-EduPack elements. The users positively mentioned the offered content of the Helper- and guided step elements in the overlays, see Figure 4.3.

[16]The recommended resolution for TVB is at least 1600x1000.

4.2.2 Task 2 – TVB-EduCase Scenario

As expected the users had more problems with the second scenario. Again they were first asked to work without TVB-EduPack.

Six out of seven were able to start a simulation in TVB, but none of them knew what they were doing. Nobody in the user group has a neuroscientific background, but all of them were very curious and interested about the software of TVB.

All users set a name for the simulation, two were directly hitting the launch button and five were changing one or more parameters without any background knowledge. Four out of this five were also using the question marks links of TVB, see Figure 3.4, but they explained afterwards that they did not find the information they were looking for. Two simulations aborted immediately after the start, due to ill-considered parameter changes.

Task 2 with TVB-EduPack

The users did better when they were allowed to use TVB-EduPack within the second scenario. They especially liked the presentation of parameters with the animations, but they still didn't understand the computational models that are used for the simulations in general. One of the users suggested creating a video describing all applicable parameters of a model in order to give a good example of the usage. Nonetheless, most of them found of them the use of TVB-EduPack helpful, especially learning about the functionality of TVB itself, and the influence of parameters in particular.

One person had troubles with TVB-EduPack because he manually completed a task before he read and fulfilled it. The problem is that the action primitives are not loaded anymore when a <*sub-step*> element is already marked as *done*.

Further suggestions regarding TVB-EduPack were:

- a seamless integration of the task into the GUI of TVB instead of presenting the task in TVB's menu, e.g. more use of speech bubble elements, similar to the used overlays
- less text, more interactive elements à learning by doing
- an option panel in TVB-EduPack showing progress and configuration options, neglect the menu with <*step*> and <*sub-step*> structure à only show current possible task and related actions
- offer an additional video for each tutorial

4.2.3 Task 3 – Create a TVB-EduPack Script

After testing the example scenario of the first prototype, I asked the three developers to extend a prepared XML script with basic elements. Therefore I described the XML scheme for tutorials, suggested the reading of the thesis's chapter 3.2 and prepared a list of accessible HTML DOM variable names of TVB's simulator page.

They were asked to create a tutorial for the simulator page that makes use of the step and sub-step structure levels. Moreover they were asked to integrate a precondition, an overlay, at least one action primitives and a description for the created tasks.

The study showed that it requires a training period and help to get started writing tutorials by oneself. All three persons were doing better when they were offered more examples, which they could use and adapt for their own interests. The challenge faced was in regards to the utilization of the action primitives. They comprise too many different attributes and are therefore prone to errors. Although all persons were able to integrate <*step*> and <*sub-step*> elements, they had problems with the obligatory elements like non-set IDs, or types, at least once. As a result they were struggling with a nonfunctional TVB-EduPack module because a debugging tool has yet to exist.

Fortunately the developers were motivated and kept working until they succeeded and were able start their own short tutorial. The resulting feedback was very helpful, especially in foresight of future developments.

5 Conclusion and Outlook

"If the human brain were so simple that we could understand it, we would be so simple that we couldn't."
Emerson M. Pugh, 1977

5.1 Contribution

With the development of the first prototype of TVB-EduPack most of the defined requirements are achieved. The first prototype is a flexible tool that facilitates users to work with an interactive assistant in TVB and empowers more advanced users to create additional tutorials using the developed XML scripting interface. Furthermore, the users are not only able to work with the presented functionalities but can also extend the scripts with further functionalities and primitives. The source code comes with an additional documentation and the already integrated functionalities and primitives are a good introduction to the application.

The first prototype provides introduction tutorials on how to import own user datasets, project management and information about how to start a simulation with this datasets. Additionally a TVB-EduCase for a parameter exploration simulation is provided that also comprises a TVB project file import technique. With the latter TVB-EduCase example the users can get a first impression of the possibilities that will arise with the development of a TVB-EduCase library. The exchange of datasets and simulation results with other workgroups will especially improve the user's experience within TVB with a lasting effect.

5.2 Outlook

With respect to my own experience and feedback from colleagues and the conducted user-study, TVB-EduPack already offers an easier introduction and help with the first prototype. The direct integration into the software of TVB helps the user more effectively than video tutorials or text-written examples. Even though I created introductory video tutorials for TVB presentations and workshops myself, the advantages of an interactive guide on the personal computer outweigh even live demonstrations on a stage. According to the workshop TVB:Node#1 – *Practical brain network modeling* that was held in Hamburg in June 2014 – the feedback about the mention of the interactive learning

platform TVB-EduPack generated a very positive response. Consequently, the possibility that TVB-EduPack will be integrated into a next release of The Virtual Brain is very good.

There are many open possibilities and opportunities to enhance the work with The Virtual Brain and especially with TVB-EduPack in the future. This outlook is based on previous set requirements and feedback from the user-study.

1. **TVB-EduCase Library**
   The TVB-EduCase Library is one of the important projects that needs to be integrated into TVB-EduPack. TVB-EduCase Library shall provide a dictionary of dynamical regimes, pre-set parameter settings from published models and a variety of brain states as well as pathologies and malfunctions. These are for example, regimes that generate resting-state activity or epileptic brain dynamics. This will allow users to get a quick overview of the dynamic repertoire of TVB for chosen parameter settings and could even be presented interactive if a wide range of parameters was explored and simulated before.
2. **TVB-Autopilot / Live Application Control**
   Although the first primitives for this tool are already integrated in the first prototype, there are a lot of possibilities with the implementation of a live application control. For example short demonstrations of TVB can be used to arouse interest for new users or to present basic functionalities of TVB without operating it manually.
3. **Editor for XML based tutorials and TVB-EduCases**
   It was the project's intention to make use of an XML-based format to have a simplified and flexible approach for the process of creating tutorials. Regarding the users' feedback – it is not as easy as expected yet.

With the help of an intelligent XML editor that only allows the usage of tutorial features and elements at positions where they fit in the XML scheme, the effort will become easier, e.g. with a graphical drag and drop editor.
Consequently potential XML content- and structure errors would be minimized and with the help of an additional debugging toolkit for developers, the tutorial authoring process could improve significantly in future.

5.3 Conclusion

The first prototype of TVB-EduPack is moving in the right direction. Although a lot of improvements need to be developed for future versions, it is already possible to help users – particularly inexperienced users.

In summary, TVB-EduPack comprises guided tutorials and Helper elements on the GUI that helps users to get started with the graphical user interface and work with TVB. The

focus of this thesis was to develop an extendable interface that facilitates users to create more and better tutorials in the future. In addition to TVB-EduCase library, an easy to handle tutorial editor that works by drag and drop would also simplify the process of creating tutorials and also improve the usability and functionality of TVB-EduPack.

At this point I consider that my contribution of TVB-EduPack and the scripting interface can open a lot possibilities regarding the creation and broadcasting of knowledge among The Virtual Brain users. The latest progress of TVB-EduPack will be presented at the *Society for NeuroScience's* annual meeting in November 2014 [16].

Bibliography

[1] K. Amunts, M. J. Hawrylycz, D. C. Van Essen, J. D. Van Horn, N. Harel, J. B. Poline, F. De Martino, J. G. Bjaalie, G. Dehaene-Lambertz, S. Dehaene, P. Valdes-Sosa, B. Thirion, K. Zilles, S. L. Hill, M. B. Abrams, P. A. Tass, W. Vanduffel, A. C. Evans, and S. B. Eickhoff. Interoperable atlases of the human brain. *NeuroImage*, (0).

[2] Korbinian Brodmann. *Vergleichende Lokalisationslehre der Grosshirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Barth, 1909.

[3] David Wallace Croft. Intelligent software agents: definitions and applications, 1997.

[4] Hugo de Garis, Chen Shuo, Ben Goertzel, and Lian Ruiting. A world survey of artificial brain projects, part i: Large-scale brain simulations. *Neurocomputing*, 74(1–3):3–29, 2010.

[5] S. B. Eickhoff, K. E. Stephan, H. Mohlberg, C. Grefkes, G. R. Fink, K. Amunts, and K. Zilles. A new spm toolbox for combining probabilistic cytoarchitectonic maps and functional imaging data. *Neuroimage*, 25(4):1325–35, 2005.

[6] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–5, 2012.

[7] Ben Goertzel, Ruiting Lian, Itamar Arel, Hugo de Garis, and Shuo Chen. A world survey of artificial brain projects, part ii: Biologically inspired cognitive architectures. *Neurocomputing*, 74(1–3):30–49, 2010.

[8] J. Goni, M. P. van den Heuvel, A. Avena-Koenigsberger, N. Velez de Mendizabal, R. F. Betzel, A. Griffa, P. Hagmann, B. Corominas-Murtra, J. P. Thiran, and O. Sporns. Resting-brain functional connectivity predicted by analytic measures of network communication. *Proc Natl Acad Sci U S A*, 111(2):833–8, 2014.

[9] Ian Hickson. Web storage, 2013.

[10] Philippe Le Hégaret. Document object model, 2005.

[11] V. K. Jirsa, O. Sporns, M. Breakspear, G. Deco, and A. R. McIntosh. Towards the virtual brain: network modeling of the intact and the damaged brain. *Arch Ital Biol*, 148(3):189–205, 2010.

[12] Viktor K Jirsa. Neural field dynamics with local and global connectivity and time delay. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1891):1131–1143, 2009.

[13] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301, 2010.

[14] Karl M Kapp. *The gamification of learning and instruction: game-based methods and strategies for training and education*. John Wiley and Sons, 2012.

[15] C. K. Machens. Building the human brain. *Science*, 338(6111):1156–7, 2012.

[16] H. Matzke, P. Triebkorn, and P. Ritter. Tvb-edupack - an interactive learning and scripting tool for the virtual brain. 10/2014 in prep.

[17] H. Matzke, P. Triebkorn, M. Schirner, and P. Ritter. An interactive learning platform for the virtual brain, 2014.

[18] Erica Naone. Tr10: Intelligent software assistant, 2009.

[19] Jenny Paay. Learning by interacting: Comparing the effectiveness of an interactive tutorial with a standard electronic book interface. 2000.

[20] Liam Quin. Extensible markup language, 2014.

[21] P. Ritter, M. Schirner, A. R. McIntosh, and V. K. Jirsa. The virtual brain integrates computational modeling and multimodal neuroimaging. *Brain Connect*, 3(2):121–45, 2013.

[22] Khaled Sabry and Jeff Barker. Dynamic interactive learning systems. *Innovations in Education and Teaching International*, 46(2):185–197, 2009.

[23] P. Sanz Leon, S. A. Knock, M. M. Woodman, L. Domide, J. Mersmann, A. R. McIntosh, and V. Jirsa. The virtual brain: a simulator of primate brain network dynamics. *Front Neuroinform*, 7:10, 2013.

[24] M. Schirner, S. Rothmeier, V. K. Jirsa, A. R. McIntosh, and P. Ritter. Constructing subject-specific virtual brains from multimodal neuroimaging. 2014 in prep.

[25] TVB-Developer's-Reference. Tvb developer's reference manual 1.0. Report, 01.09.2014 2014.

[26] TVB-User-Guide. Tvb user guide 1.0. Report, 01.09.2014 2014.

[27] Martijn P. van den Heuvel and Hilleke E. Hulshoff Pol. Exploring the brain network: A review on resting-state fmri functional connectivity. *European Neuropsychopharmacology*, 20(8):519–534, 2010.

[28] M Marmaduke Woodman, Laurent Pezard, Lia Domide, Stuart A Knock, Paula Sanz-Leon, Jochen Mersmann, Anthony R McIntosh, and Viktor Jirsa. Integrating neuroinformatics tools in the virtual brain. *Frontiers in neuroinformatics*, 8, 2014.

[29] Dongsong Zhang. Interactive multimedia-based e-learning: A study of effectiveness. *American Journal of Distance Education*, 19(3):149–162, 2005.

[30] Dongsong Zhang, Lina Zhou, Robert O. Briggs, and Jay F. Nunamaker Jr. Instructional video in e-learning: Assessing the impact of interactive video on learning effectiveness. *Information and Management*, 43(1):15–27, 2006.

6 Appendix

"Beware of bugs in the above code; I have only proved it correct, not tried it."
Donald Knuth

6.1 Complete XML Structure Listing of Action Primitives

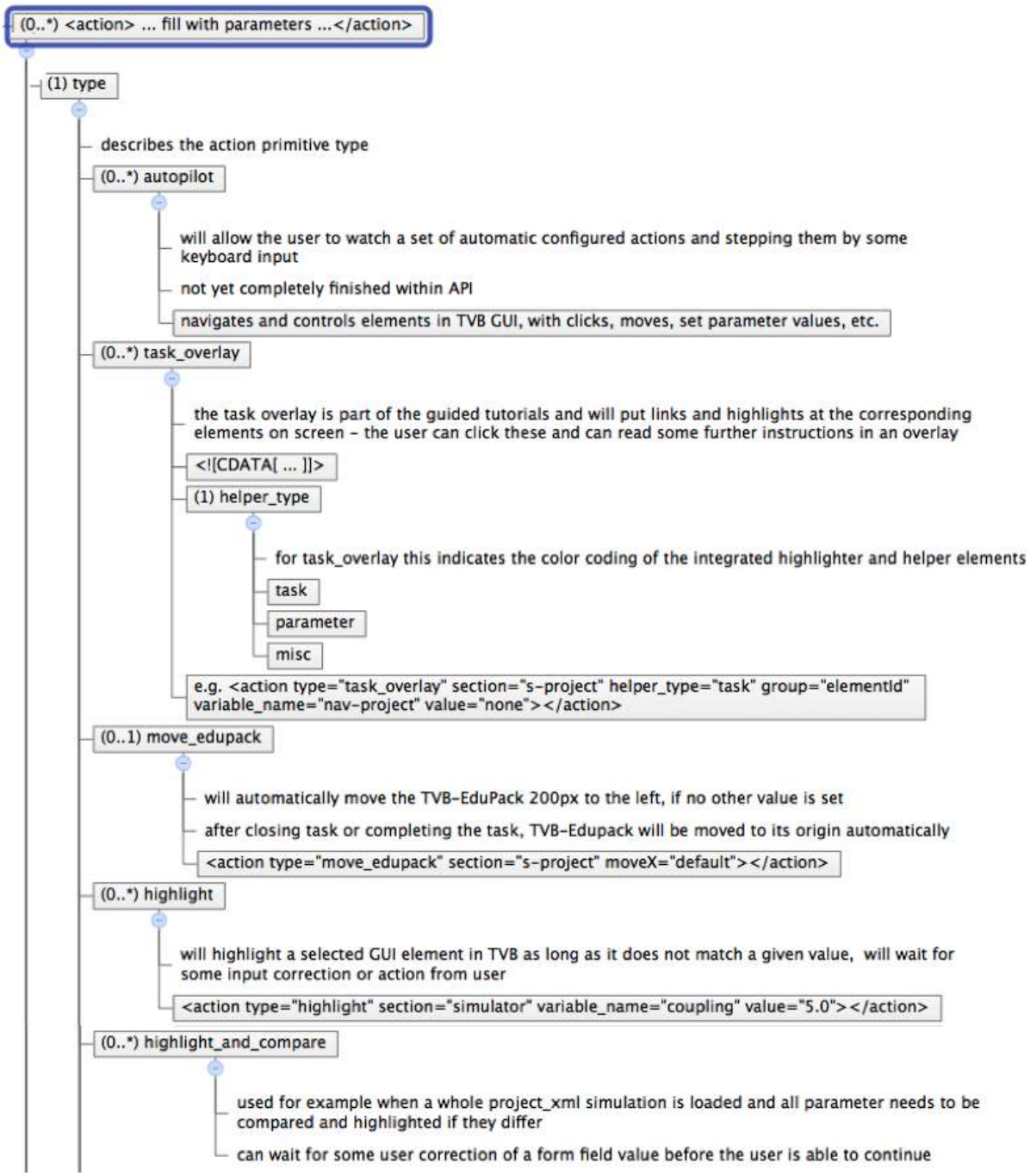

Figure 6.1: Part 1 of Complete XML Structure Listing of Action Primitives.

Figure 6.2: Part 2 of Complete XML Structure Listing of Action Primitives.

6.2 Complete XML Tutorial File for the Use-Case

Listing 6.1: Complete XML Tutorial File for the Use-Case

```
<?xml version="1.0" encoding="utf-8"?>
<edupack_tutorial>
  <!-- title for this tutorial and tutorial_name for internal reasons e.g. preconditions -->
  <title>Thesis Example Tutorial, TVB-EduPack v0.2</title>
  <tutorial_name>tutorial_thesis</tutorial_name>

  <!-- preconditions for whole tutorial, here deactivated -->
  <preconditions>
    <!-- <state>finished tutorial_import</state> -->
  </preconditions>

  <!-- description text above menu -->
  <description>
    <![CDATA[
      Welcome to the brief thesis example tutorial.<br>
      The exercise is to <b>create a new project</b> and then <b>start a simple simulation</b> with
      specified parameters and the usage of default datasets. Therefore the tutorial is divided in
      two parts parts. <br>
      It will not focus on details but combine some of the presented action primitives.
    ]]>
  </description>

  <!-- here starts the first step with all its sublevel structure elements -->
  <step>
    <label>Create a new project</label>
    <step-id>00</step-id>

    <!-- here starts the first sub-step, no preconditions but using task_overlay primitive
      to emphasize the task - here to switch into the Project page of TVB -->
    <sub-step>
      <task>Switch to project interface</task>

      <!-- type is "description", later in the tutorial it will change to "parameter" -->
      <type>description</type>

      <!-- sub-step id -->
      <value>01</value>

      <!-- description of the task -->
      <detail>
        <![CDATA[
          To create a new project and not only a new simulation, we need to
          change to the project page of TVB.<br/>
          Therefore, please click the
          <img src=/images/icons_green11.png width=5% valign=bottom>
          <b>Project button</b>
          <img src=/images/icons_green11_2.png width=5% valign=bottom>
          in the footer menu.
        ]]>
      </detail>

      <!-- action primitive "task_overlay", showing a numbered Helper element
        (number of the current sub-step) and an additional message presented in an overlay.-->
      <action type="task_overlay" section="s-project" helper_type="task"
              group="elementId" variable_name="nav-project" read="true">

        <![CDATA[
          This is the first task you have to solve, <br>
          click the <b>Project</b> button to switch to the Project page.<br><br>
          <img src=/images/tutorial_preview_project.png width=400>
        ]]>
```

```
62      </action>
63
64      <!-- no auto completion within this tutorial, if the user clicks the link, the next substep
65        will be opened and the previous task will be marked as finished -->
66      <finish_constraints>
67        <manual>Click here, when you completed the task.</manual>
68        <!-- <state>none</state> -->
69      </finish_constraints>
70    </sub-step>
71
72    <sub-step>
73      <task>Go to List of All Projects</task>
74      <type>description</type>
75      <value>02</value>
76
77      <!-- related to the previous task, the user should now be on the project page,
78        otherwise he / she will get a warning message and is not able to open the sub-step -->
79      <precondition section="s-project">
80        <![CDATA[
81        <font color=red>Warning</font>, you are not on the project page. <br>
82        Please switch to the Project page in order to continue.
83        ]]>
84      </precondition>
85
86      <action type="task_overlay" section="s-project" helper_type="task"
87            group="elementId" variable_name="nav-project-list">
88        <![CDATA[
89        The menu entry <b>List of all projects</b> will bring you to the project manager,
90        <br>where we can generate a new and clean project.
91        ]]>
92      </action>
93
94      <detail>
95        <![CDATA[
96        Now you see all submenu options of TVB's project page. <br/>
97        We want to create a new project, so please click the button <br/>
98        <img src=/images/icons_green11.png width=5% valign=bottom>
99        <b>List of all projects</b>
100       <img src=/images/icons_green11_2.png width=5% valign=bottom> to continue.<br/>
101       ]]>
102     </detail>
103
104     <finish_constraints>
105       <manual>Click here, when you completed the task.</manual>
106       <!-- <state>none</state> -->
107     </finish_constraints>
108   </sub-step>
109
110   <sub-step>
111     <task>Create a new Project</task>
112     <type>description</type>
113     <value>03</value>
114
115     <precondition section="s-project">
116       <![CDATA[
117       <font color=red>Warning</font>, you are not on the correct page
118       (Project -> List of all Projects)!
119       ]]>
120     </precondition>
121
122     <detail>
123       <![CDATA[
124       Now you are in the project manager of TVB,it shows all of your
125       created projects including open, finished or aborted operations.<br>
126       As probably recognized, the TVB-EduPack moved a little bit to the left.<br/>
127       Right to this window is the button
```

```
128         <img src=/images/icons_green11.png width=5% valign=bottom>
129         <b>Create new Project</b>
130         <img src=/images/icons_green11_2.png width=5% valign=bottom>
131         , click this and continue.
132         ]]>
133       </detail>
134
135       <action type="task_overlay" section="s-project" helper_type="task"
136               group="elementName" variable_name="create">
137         <![CDATA[
138         Click <b>Create new project</b> and then we will create a new project.
139         ]]>
140       </action>
141
142       <!-- action primitive to move TVB-EduPack some pixels to the left
143       in order to uncover hidden elements -->
144       <action type="move_edupack" section="s-project">
145       </action>
146
147       <finish_constraints>
148         <manual>Click here, when you completed the task.</manual>
149         <!-- <state>none</state> -->
150       </finish_constraints>
151     </sub-step>
152
153     <sub-step>
154       <task>Save new project</task>
155       <type>description</type>
156       <value>04</value>
157
158       <precondition section="s-project">
159         <![CDATA[
160         <font color=red>Warning</font>, you are not in the correct interface
161         (Project -> Create New Project)!
162         ]]>
163       </precondition>
164
165       <detail>
166         <![CDATA[
167         Give your project<br/>
168         <img src=/images/icons_green11.png width=5% valign=bottom>
169         <b>a name and a description</b>
170         <img src=/images/icons_green11_2.png width=5% valign=bottom>
171         and then continue by clicking<br/>
172         <img src=/images/icons_green11.png width=5% valign=bottom>
173         <b>Save Changes</b>
174         <img src=/images/icons_green11_2.png width=5% valign=bottom><br/>
175         right next to the TVB-EduPack window.
176         ]]>
177       </detail>
178
179       <!-- three task_overlay elements to relate several
180       fields and buttons to this task -->
181       <action type="task_overlay" section="s-project" helper_type="task"
182               group="elementId" variable_name="name">
183         <![CDATA[ Please give your project a significant <b> Name</b>.]]>
184       </action>
185
186       <action type="task_overlay" section="s-project" helper_type="task"
187               group="elementId" variable_name="description">
188         <![CDATA[ Please give your project a meaningful <b> Description</b>. ]]>
189       </action>
190
191       <action type="task_overlay" section="s-project" helper_type="task"
192               group="elementName" variable_name="save" read="true">
193         <![CDATA[ When the project is named, you can
```

```
194         continue and <b>Save Changes</b>. ]]>
195       </action>
196
197       <action type="move_edupack" section="s-project">
198       </action>
199
200       <finish_constraints>
201         <manual>Click here, when you completed the task.</manual>
202         <!-- <state>none</state> -->
203       </finish_constraints>
204     </sub-step>
205
206     <sub-step>
207       <task>Congratulations</task>
208       <type>description</type>
209       <value>05</value>
210
211       <detail>
212         <![CDATA[
213         <b>Congratulation</b>, you created a clean project and are now ready to start
214         a simple simulation. Make sure your project is selected, indicated with a
215         dark star in the upper left area. <br>
216         Now switch to
217         <img src=/images/icons_green11.png width=5% valign=bottom>
218         <b>TVB simulator page</b>
219         <img src=/images/icons_green11_2.png width=5% valign=bottom> on the left.<br/>
220         , where we will set up your first simulation.
221         ]]>
222       </detail>
223
224       <action type="task_overlay" section="s-project" helper_type="task"
225               group="elementId" variable_name="nav-burst" value="none" read="true">
226
227         <![CDATA[
228           To continue with the next task and create your first simulation, <br>
229           click the <b>Simulator</b> button to switch to the Simulator page.<br><br>
230           <img src=/images/tutorial_preview_simulator.png width=400>
231         ]]>
232       </action>
233
234       <finish_constraints>
235         <manual>Click here, when you completed the task.</manual>
236         <!-- <state>none</state> -->
237       </finish_constraints>
238     </sub-step>
239
240   </step>
241
242   <step>
243     <label>Start your first simulation</label>
244     <step-id>01</step-id>
245
246     <sub-step>
247       <task>Create a new simulation</task>
248       <type>parameter</type>
249       <value>01</value>
250
251       <precondition section="s-burst">
252         <![CDATA[
253         <font color=red>Warning</font>, you are not on TVB Simulator page,
254         please change in order to continue.
255         ]]>
256       </precondition>
257
258       <detail>
259         <![CDATA[
```

```
260         There are two ways how you can start a simulation - either you start a new
261         one and adapt the default parameter values or you copy one of your old
262         simulations from history. Your simulation history is empty, so we will
263         start to create a simulation.<br>
264         Click on
265         <img src=/images/icons_green11.png width=5% valign=bottom>
266         <b>the star in the history panel</b>
267         <img src=/images/icons_green11_2.png width=5% valign=bottom><br/>
268         to create a new simulation.
269         ]]>
270       </detail>
271
272       <!-- burst-history link -->
273       <action type="task_overlay" section="s-burst" helper_type="task"
274               group="elementClass" variable_name="view-history view-column col-1"
275               read="true">
276
277         <![CDATA[
278           Clicking this button creates a new and clean simulation.<br>
279           You can skip this task, if you see an empty form field next to
280           the <b>New simulation core</b>. <br>This means that you are already
281           ready for the next task. :)<br><br>
282           <img src=/images/tutorial_preview_simulator_name.png width=400>
283         ]]>
284       </action>
285
286       <finish_constraints>
287         <manual>Click here, when you completed the task.</manual>
288         <!-- <state>none</state> -->
289       </finish_constraints>
290     </sub-step>
291
292     <sub-step>
293       <task>Start to configure your simulation</task>
294       <type>parameter</type>
295       <value>02</value>
296
297       <precondition section="s-burst">
298         <![CDATA[
299         <font color=red>Warning</font>, you are not on TVB Simulator page,
300         please change in order to continue.
301         ]]>
302       </precondition>
303
304       <detail>
305         <![CDATA[
306         At first, give your simulation a
307         <img src=/images/icons_green11.png width=5% valign=bottom>
308         <b>name</b>
309         <img src=/images/icons_green11_2.png width=5% valign=bottom> and then continue
310         by having a first look at the parameters below.<br/>
311         Please enable the TVB-Parameter Helper elements from the EduPack GUI and use the
312         slider to decrease the visibility if they are distracting you.<br>
313         Open the Help content and watch the animations to get an idea of the
314         first three parameters.
315         ]]>
316       </detail>
317
318       <action type="task_overlay" section="s-burst" helper_type="task"
319               group="elementId" variable_name="input-burst-name-id" read="true">
320
321         <![CDATA[
322           Give your simulation a significant name and learn about
323           TVB-EduPack Helper elements by activating the help for the parameters.<br><br>
324           <img src=/images/tutorial_preview_helper.png width=400>
325         ]]>
```

```
326       </action>
327
328       <!-- example for highlight primitive, with defined and undefined value-to-be -->
329       <action type="highlight" section="simulator"
330               variable_name="conduction_speed" value="40.0"></action>
331       <action type="highlight" section="simulator"
332               variable_name="coupling_parameters_option_Linear_b"></action>
333
334       <finish_constraints>
335         <manual>Click here, when you completed the task.</manual>
336         <!-- <state>none</state> -->
337       </finish_constraints>
338     </sub-step>
339
340     <sub-step>
341       <task>Preview results and adapt parameters</task>
342       <type>parameter</type>
343       <value>03</value>
344
345       <precondition section="s-burst">
346         <![CDATA[
347         <font color=red>Warning</font>, you are not on TVB Simulator page,
348         please change in order to continue.
349         ]]>
350       </precondition>
351
352       <detail>
353         <![CDATA[
354         Now that you have the first impression of a small set of parameters,
355         start this overlay here which will show you some results of
356         previously computed simulations. <br>
357         Use the slider elements to compare the results and select one
358         configuration that you want to submit directly to TVB-Simulator page.<br>
359         Clicke here to open the overlay:
360         ]]>
361       </detail>
362
363       <overlay>
364         <content>
365         <![CDATA[
366           <h2>Compare Neuronal activity with Power Spectrum and FC Matrices</h2>
367           <br>
368           Showing FC and PSD of Parameter Exploration of the parameters linear coupling value a and the
369           conduction speed in order to get a good neural activity.<br> User the two different result
370           frames to compare the data of different parameter ranges and submit the one you like most
371           to the TVB Simulator Interface.
372           <br><br><hr>
373             <table align=center width=100%>
374               <tr width=100%>
375                 <td width=50% align=left class=rightborder>
376                     <img width=100% id=img_A src=/images/sim_images/
377                       pics_5s_subs_d0.75_1.0_2.5_a0.04_0.01_0.2_cs20_20_60_fc/d_2_5/c_40/
                              timeline_plotter_15.jpg>
378                 </td>
379                 <td width=50% align=left>
380                     <img width=100% id=img_B src=/images/sim_images/
381                       pics_5s_subs_d0.75_1.0_2.5_a0.04_0.01_0.2_cs20_20_60_fc/d_2_5/c_40/
                              timeline_plotter_15.jpg>
382                 </td>
383               </tr>
384               <tr>
385                 <td class=rightborder>
386                   <table align=center width = 80% >
387                   <tr width=80%>
388                     <td width=90% align=left>
389                       Long-range coupling factor<br/>
```

```
                    <input
                      name=coupling_A
                      type=range
                      value=01
                      max=31
                      min=01
                      id=coupling_A
                      class=overlaySlide
                      oninput=slider_A()
                      onchange=slider_A() >
                  </td>
                  <td align=right>
                    <output
                        name=val_coupling_A
                        id=val_coupling_A
                        for=coupling_A>0.110</output>
                  </td>
                </tr>
                <tr>
                  <td align=left>
                    Conduction Speed<br/>
                    <input
                      name=speed_A
                      type=range
                      value=2
                      max=3
                      min=1
                      id=speed_A
                      class=overlaySlide
                      oninput=slider_A()
                      onchange=slider_A()>
                  </td>
                  <td align=right>
                    <output
                        name=val_speed_A
                        id=val_speed_A
                        for=speed_A>40</output>
                  </td>
                </tr>
                <tr>
                  <td align=left><br><br>
                    <b>Standard TVB Parameters, except:</b><br>
                    Local Dynamic Model: Stefanescu Jirsa 3D<br>
                    Integrator: HeunStochastic,
                    Noise D: 2.5<br>
                    monitors: subsample (5ms)<br>
                    simulation length: 5 seconds
                  </td>
                  <td>
                    <p align=right><a href=# class=submit_button
                      onclick=fwdSimLengthToSimulator(5000); fwdModelToSimulator(sj3d);
                      fwdIntegratorToSimulator(); fwdMonitorsToSimulator();
                      fwdSubSampleTimeToSimulator(5); fwdNoiseToSimulator(2.5);
                      fwdCouplingToSimulator(val_coupling_A); fwdSpeedToSimulator(val_speed_A);
                      closeThisOverlay()>Submit Data</a></p>
                  </td>
                </tr>
              </table>
              </td>
              <td>
                <table align=center width = 80% >
                <tr width=80%>
                  <td width=90% align=left>
                    Long-range coupling factor<br/>
                    <input
                      name=coupling_B
```

```
                    type=range
                    value=01
                    max=31
                    min=01
                    id=coupling_B
                    class=overlaySlide
                    oninput=slider_B()
                    onchange=slider_B() >
                </td>
                <td align=right>
                  <output
                      name=val_coupling_B
                      id=val_coupling_B
                      for=coupling_B>0.110</output>
                </td>
              </tr>

              <tr>
                <td align=left>
                  Conduction Speed<br/>
                  <input
                    name=speed_B
                    type=range
                    value=2
                    max=3
                    min=1
                    id=speed_B
                    class=overlaySlide
                    oninput=slider_B()
                    onchange=slider_B()>
                </td>
                <td align=right>
                  <output
                      name=val_speed_B
                      id=val_speed_B
                      for=speed_B>40</output>
                </td>
              </tr>
              <tr>
                <td align=left><br><br>
                  <b>Standard TVB Parameters, except:</b><br>
                  Local Dynamic Model: Stefanescu Jirsa 3D<br>
                  Integrator: HeunStochastic,
                  Noise D: 2.5<br>
                  monitors: subsample (5ms)<br>
                  simulation length: 5 seconds
                </td>
                <td>
                  <p align=right>
                    <a href=# class=submit_button
                      onclick=fwdSimLengthToSimulator(5000); fwdModelToSimulator(sj3d);
                      fwdIntegratorToSimulator(); fwdMonitorsToSimulator();
                      fwdSubSampleTimeToSimulator(5); fwdNoiseToSimulator(2.5);
                      fwdCouplingToSimulator(val_coupling_B); fwdSpeedToSimulator(val_speed_B);
                      closeThisOverlay()>Submit Data</a></p>
                </td>
              </tr>
            </table>
          </td>
          </tr>
        </table>
    ]]>
    </content>
  </overlay>
  <finish_constraints>
    <manual>Click here, when you completed the task.</manual>
```

```
522         <!-- <state>none</state> -->
523       </finish_constraints>
524     </sub-step>
525
526     <sub-step>
527       <task>Launch your simulation</task>
528       <type>parameter</type>
529       <value>04</value>
530       <precondition section="s-burst">
531         <![CDATA[
532         <font color=red>Warning</font>, you are not on TVB Simulator page,
533         please change in order to continue.
534         ]]>
535       </precondition>
536       <detail>
537         <![CDATA[
538         You can see now, that your selected parameters were directly transferred into
539         the TVB Simulator configuration panel. Other parameters that were used during
540         the previous simulation were also set
541         (e.g. local dynamic model, noise and simulation length).<br>
542         If you want to achieve the same computational results, click the
543         <img src=/images/icons_green11.png width=5% valign=bottom>
544         <b>Launch Simulation</b>
545         <img src=/images/icons_green11_2.png width=5% valign=bottom> button.
546         And that's it for your first simulation. :)
547         ]]>
548       </detail>
549       <action type="task_overlay" section="s-burst" helper_type="task"
550               group="elementId" variable_name="button-launch-new-burst" read="true">
551         <![CDATA[
552           Hit the launch button if you're interested to analyze the data.<br>
553           Approximated computational time: not available<br>
554           <img src=/images/tutorial_preview_run.png width=400>
555         ]]>
556       </action>
557       <finish_constraints>
558         <manual>Click here, when you completed the task.</manual>
559         <!-- <state>none</state> -->
560       </finish_constraints>
561     </sub-step>
562   </step>
563 </edupack_tutorial>
```