**EE-Browser Redesign**

**Assumptions/Ideas/SWAD**

1. **CORE FEATURES**

* Stand-alone Windows-Client software solution for explicit usage in Daimler environment
* Use of latest user control technologies like MS Ribbon and dock-able control panes
* Orthogonal and stringent usage control concept
* Reasonable system feedback in case of loading operations (progress bars, etc.)
* Support for opening archive files including one KBL and at least one SVG file
* Working on multiple opened documents simultaneously (Multi-document handling) with possibility of interactive linking between inliners
* Ability to allow loading of very large-scaled documents with efficient memory usage
* DATABASE: ‘LDorado’ KBL container file
  + Full support for a predefined KBL version
  + Deserialization of complete KBL container content with efficient access to all included information
  + Proper data organization for displaying varied information in different grids
  + Available features like module configuration or compare functionalities has to be handled directly via extracted data objects from KBL
  + Filter information in grids
  + Roundtrip functionality to highlight object dependencies
* GRAPHICAL INTERFACE: ‘LDorado’ SVG file
  + Load of complete graphic from SVG via converter process into EE-Browser (Original mode)
  + Load basic graphic form KBL via separate converter process into EE-Browser (Draft mode)
  + Possibility to identify, select and highlight (lowlight) graphical objects
  + Basic concepts for graphical modifications
  + Print (Plot) and export of document
  + 3D visibility of harness/topology drawing and visualization of 3D bundles and companion objects like connectors/fixings
* Cross-highlight behavior (data grids 🡨🡪 graphic)
* Intelligent search functionality over all document attributes
* Fully control via individual user configuration settings which information with which header captions in which order should be displayed in all grids
* Module configuration
  + Activate/Deactivate modules and their assigned data objects
  + Load/Save predefined configuration setting
  + Grey-out partly inactive figures like connectors or segments
  + Module connectivity view
* Praxis-proven Redlining functionality
  + Assign text-based comments or graphical sketches against individual data objects and their referred graphical representation
  + Save/Restore comments
* Import/Export interfaces
  + Data import from different formats (DSI, …)
  + Data export to Excel
  + Graphical export to DWG, DXG, image formats
* Compare documents
  + Compare data objects and graphical representation
  + Highlight changes
  + Export results
* Analysis views
  + Providing different analysis views causing in various graphical representations
  + Lowlight/Highlight inactive and active figures
* Navigator control
  + Small control pane displaying the document bounds used for quick navigation thru the graphic
  + Active zoom and pan functionality would be helpful
* Additional features
  + Connectivity view based on selected connector
  + Start/End connector table view based on selected wire
  + Copper content view based on selected connector
  + Navigator control with pan and zoom functionality
  + BOM view based in additional ‘Index.xml’ file

1. **ASSUMPTIONS/IDEAS**

* SOFTWARE CORE:
  + Windows-platform based client application with one main MDI-based form
  + One closed-loop solution structure bases on Framework 4 to enable TPL support to use different available processor cores for simultaneously loading of KBL and SVG
  + Usage of latest 3rd-party components (Infragistics NetAdvantage, VectorDraw, etc.)
  + User-friendly usability and look & feel of the application have to be based on actual/modern control concepts (Ribbon, icons, dock-able controls, etc.)
  + Guidance functionalities and tooltip support wherever it’s useful are desirably
  + Localization preferred or not has to be discussed (if needed the complete handlings must be implemented in an early stage)
  + Central-maintained log file mechanism for all different parts of the software with save functionality (better would be a log console control to give the user the possibility to understand occurring problems or errors)
  + Look & Feel and content of data grids should be maintained by different XML config files which will be automatically generated if they are not existing
  + Usage of Task Parallel Library for load processes
* LIBRARIES:
  + All functionalities which are elementary necessary for a proper and accurate working process with EE-Browser has to be bundled into one single solution
  + There should be no dependencies to any satellite libraries or existing ‘Intedis’ assemblies
  + Simultaneous update of versioning for all parts of the software
* PROTECTION:
  + Use of CodeMeter (known protection environment but also some pending issues regarding CmAct and license borrowing)
  + In case of using CodeMeter it makes sense to only protect a part of the software to improve startup performance of the application
  + Use of alternative licensing products (FlexLM) but in this case a lot of costs and time-wasting efforts will follow
  + Maybe possible to implement our own protection business logic (could be complicated and also time-consuming)
* FEATURE CONTROL:
  + Intelligent involvement for enabling/disabling functionalities through control switches based on a predefined feature matrix
  + Administrative mode to change core settings of EE-Browser on customer side
  + Feature control must be implemented completely independent from software protection
* DATABASE:
  + Adopt behavior of existing KBL importer and access complete KBL container via XML deserialization features available in Framework classes
  + Enhance business logic to check version information and data structure of KBL file
  + As the HCV container are stored in PDM systems there is a mandatory requirement to open files containing an older KBL version as the current defined
  + Last mentioned point makes it necessary to check compatibility and integrity of loaded KBL files during load
  + Use of automatic generated data object class structure based on KBL XSD schema definition to access all relevant information
  + All existing KBL data which are needed for the complete working processes in EE-Browser has to be loaded once when the container will be opened
  + Working directly with given objects and their attributes to fill user-defined grids (therefore an intelligent data adapter has to be generated which fills the different object type-separated grids in a reasonable way)
  + Efficient and memory-friendly implementation of object mapper based on dictionaries or other kinds of collection structures
  + Performance-optimized mapper between data objects and their graphical representation including relations between dependencies or hierarchical objects
  + Data grid visualization and interactive functionalities like filtering and roundtrip to highlight object dependencies
* MODULES:
  + There could be a chance to implement a load/reload handling which allows the user during document load to define (or load) the active module configuration which should be displayed in EE-Browser
  + This will reduce memory usage according to the filtered graphical content which is displayed in EE-Browser
  + Only SVG groups which are active in selected module configuration has to be loaded but complete KBL file
  + Change of module configuration during runtime must be work quickly
  + Has to be clarified with Daimler if this feature is desired or not
* GRAPHICAL INTERFACE:
  + Usage of **VectorDraw** component to load fully SVG graphic into EE-Browser (ORIGINAL MODE):
    - Usage of existing SVG converter process to load complete SVG graphic into EE-Browser with all details based on original exported SVG file from LDorado system
    - Enhance given convert functionality and improve memory usage of created figures (hatch properties, white fill areas, etc.)
    - Single SVG graphics are part of groups which are transformed to SVG group objects belonging to the VectorDraw document based on our own shape definition
    - SVG groups can contain other SVG group objects (hierarchical structure)
    - SVG group objects has their own attributes, render and highlight functionalities
    - Highlight/Lowlight of part graphics (other SVG groups) included in parent SVG group and vice versa handling
    - Adding redlining change flags as graphical enhancement directly to the SVG group object
    - Visualization of tooltip on graphical SVG group with basic technical information of regarding data object
    - 3D mode and conversion of flat 2D objects like segments and connector faces to bundles and boxes with predefined layout
    - Visualization of real diameter for bundles in 3D mode
    - Simplified visualization of tape layout for 3D bundles
  + Usage of **VectorDraw** component to load draft graphic into EE-Browser (DRAFT MODE):
    - Implementation of functionalities to generate basic graphical representation with vertices, segments and connector tables founded explicitly on KBL data
    - Allows quickly opening very large-scaled documents in EE-Browser
    - Switching between draft and original mode for currently opened document
    - Selection and highlight/lowlight functionalities similar to the original mode
    - Display connector tables with basic information of assigned cavities and wiring
    - Allow user-defined modifications of graphical figure appearance through predefined configuration settings in an XML file
    - 3D visualization should be also available in draft mode
  + Usage of own programmed **GDI+ graphical interface** to load fully SVG graphic into EE-Browser (ORIGINAL MODE):
    - Implementation of a own graphical interface aimed to exactly fits the necessary requirements for loading a huge number or complex graphical objects from a SVG file
    - Use of Framework classes to draw basic graphical shapes like their graphical definition with all attributes based on the SVG file
    - Need to create own business logic for a suitable data model to store group and hierarchical structures with needed attributes
    - Defining handlings for rendering (highlight/lowlight) graphical objects
    - Identification and selection algorithm for graphical objects which belongs together (SVG groups)
    - View/Zoom and refresh handlings has to be handled by ourselves
    - Realization for converting flat 2D drawing to 3D graphic
  + Usage of own programmed **GDI+ graphical interface** to load partly graphic into EE-Browser (DRAFT MODE):
    - Implementation of a own graphical interface for generating graphical figures based on KBL data (similar representation like draft mode described for VectorDraw component)
    - Memory-saving and rapid solution to display complete car topologies
    - Usage of Framework APIs to draw predefined graphical shapes defined by attributes located inside the KBL file
    - Possibility to modify graphical appearance of all figures (XML definition)
    - 3D visualization of bundles
    - Rendering, selection behavior and highlight functions identical to the original mode
  + Usage of **SVG object model** to implement an efficient reload mechanism depending on view size (zoom level) and visible area of complete drawing:
    - Implementation of a concept to prevent load of complete SVG graphic with all figures at once to reduce memory usage and prevent memory overflows
    - Graphical handlings in collaboration with VectorDraw component
    - Possibility to read all SVG elements into a well-formatted object class structure with easy access and identification of each SVG figure/group
    - Based on the SVG object model and the zoom level and visible part of the hole document a more and fewer detailed snapshot (a generated picture drawn to the render) could be presented
    - The more the view size will be reduced (go deeper inside the graphic), the more detailed the graphical artifacts has to be drawn and vice versa
    - Automatic reload steps must be defined in a configurable setting or could be triggered by user
    - Predefined appearance settings of simple figure objects which are not displayed in original definition (bounding, filling, detail level)
    - For each object type a fix number of detail levels and the visible appearance has to be defined
    - Selection, highlight and lowlight as well as analysis functionalities must be worked as described in other possible approaches
    - Fast jump between different parts of the drawing
  + Print and plot functionalities for all available document states in a user-friendly way
  + Export functionalities (DWG, DXF, etc.) for all available document states
* PLUGIN INTERFACE:
  + Ability to load assemblies located in the installation directory of EE-Browser via a predefined interface and configuration settings
  + Display of integrated (loaded) assemblies in own menu
  + Could be necessary for additional analysis views or customer-specific add-on functionalities
* COMPANION FEATURES:
  + Analysis views with different filter possibilities (orthogonal basic graphical behavior)
  + Additional lists and views (connectivity view based on selected connector or module, start/end connectors of selected wire, etc.)
  + Wire route tracing (highlight wire routes and assigned cavities, lowlight rest)
  + Interconnection between simultaneously opened documents which belongs together
  + Allow usage of pluggable modules/assemblies located in the installation folder of EE-Browser in a separate menu like in HARcad
  + Powerful compare functionalities between different version of the same documents maybe also for graphical parts (therefore some more requirements form customers side are needed to improve existing functionality)
* DOWNWARD COMPATIBILITY ISSUES:
  + We should keep in mind that already converted HCV container cannot be used in the new redesign of EE-Browser any more
  + The reason for that is the ID mapping information on the SVGGroup elements in the generated VDCL drawings
  + In the current existing solution a GUID which comes from the converted XML (our own XML HCV data model) is used to establish the link between graphical representation and data object
  + A update conversion functionality may be implemented to check if HCVs needs to be updated to new format (this means remove XML and VDCL files and regenerate VDCL files)

1. **SWAD**

|  |  |  |  |
| --- | --- | --- | --- |
| **GRAPHICAL INTERFACE:** Usage of **VectorDraw** component to load fully SVG graphic into EE-Browser (ORIGINAL MODE) | | | |
| **Strengthen** | **Weaknesses** | **Advantages** | **Disadvantages** |
| * Known and powerful CAD plugin * Object selection, identification and rendering handled by VectorDraw * Easy-to-use for graphical modifications | * Costs a lot of memory usage to convert/display full SVG to CAD drawing * A lot of CAD functionalities and object properties are not needed | * Overtake algorithms from previous SVG convert solution * Known behavior and implementation issues * Fluent and quick navigation once drawing is loaded | * Too many memory will be used for displaying a huge SVG * Hatch property handling could be improved * A lot of different integrated collections and objects are unnecessary for our case |

RESULTS AFTER TESTS: This solution is to be preferred. If the underlying software architecture works like tested in a POC we should not run into memory overflow exceptions and are able to load more than one huge document into EE-Browser. If preparing for update to latest VectorDraw version some performance and memory usage issues are currently improved by the Greece guys. The key to success is the usage of the correct processor architecture and a lean and improved conversion process from SVG to the VDCL file format. Maybe some enhancements have to be done in the actual version of the converter process.

|  |  |  |  |
| --- | --- | --- | --- |
| **GRAPHICAL INTERFACE:** Usage of **VectorDraw** component to load draft graphic into EE-Browser (DRAFT MODE) | | | |
| **Strengthen** | **Weaknesses** | **Advantages** | **Disadvantages** |
| * Known and powerful CAD plugin * Reduce graphic to elementary object types like vertices, segments and connector tables * Allows very quick loading of complete car topologies with minimal memory use | * Missing of additional graphical “landmarks” could handicap user while navigating through the document * Use of own configuration structure to define layout and content for connector tables | * Simple draft view for display several documents in one session * Quick load and fast navigation * Only important objects are displayed | * Hard to identify current location in case of huge topology drawings * Missing graphical elements could be needed for further investigation of drawing * Switch between draft/original mode can cost some time |

RESULTS AFTER TESTS: This solution is reasonable. There could be implementation of an efficient algorithm extract all necessary data outside of the KBL file combined with some logic from EEcad to use custom VectorDraw object for all drawing objects and generating connector tables in an easy way. Not much memory will be used and a quickly navigation thru the document is guaranteed. Load times will be minimized, a change between this and the original mode should be available.

|  |  |  |  |
| --- | --- | --- | --- |
| **GRAPHICAL INTERFACE:** Usage of own programmed **GDI+ graphical interface** to load fully SVG graphic into EE-Browser (ORIGINAL/DRAFT MODE) | | | |
| **Strengthen** | **Weaknesses** | **Advantages** | **Disadvantages** |
| * Handle all necessary graphical objects by our own * Using only necessary properties * No mismatch between original SVG content and displayed drawing because of using exactly the graphical information stored inside of the SVG | * Either using of 3rd party framework to load and handle SVG graphics or implement an own interface * Selection, object identification and highlight handlings not so convenient like CAD-based drawings * Render logic has to be handled manually | * Reduced usage of memory in case of loading only elementary graphics read out of the original SVG file * Quick load and fast navigation * Full control of every aspect of the graphical representation | * Update/Render handling could make it unusable in case the refresh time span is too high * Complete maintenance of drawing has to be handled by our own * More difficult and time-consuming implementation |

RESULTS AFTER TESTS: SVG loaded into a panel control and basic logic used from SVG DOM and render class library based on C# costs a lot of memory usage as well (much more than only displayed in IE and also much more than expected). The navigation/scroll/zoom handlings are very slow and unhandy. It is definitely not recommended for loading very huge drawings.

|  |  |  |  |
| --- | --- | --- | --- |
| **GRAPHICAL INTERFACE:** Usage of **SVG object model** to implement an efficient reload mechanism depending on view size (zoom level) and visible area of complete drawing | | | |
| **Strengthen** | **Weaknesses** | **Advantages** | **Disadvantages** |
| * Use VectorDraw in relation of the preloaded SVG DOM to simplify graphical content depending on visible view area and zoom level * Allows to reduce memory usage and handle multiple opened document simultaneously | * Hard to find out which kind of graphical objects has to be drawn in original version and which not * Additional configurations are needed to define layout and behavior of simplified objects * Refresh and navigation could be lame | * Reduce necessary usage of memory and ensure loading of large-scaled topology drawings as we try to simplify most of the complex detailed graphical figures * Only the visible area of the drawing will be loaded and displayed in detail | * Difficult and complex algorithms are needed to implement a reasonable solution to handle all upcoming update, render and navigation issues * The success of saving a lot of memory is not finally guaranteed because of loading the complex SVG DOM * Could slows-down all user interactions on drawing |

RESULTS AFTER TESTS: There is a chance to load only the DOM of the SVG document into memory (costs only a few MBs) and access in a convenient way the different SVG elements. But the rest of the implementation is trickier. Means we have to identify all visible objects in a specific view area, setup a lot of predefined graphical information to draw dummy objects and implement a lot of stuff to detail the drawing the more the user zooms in. Looks like a very hard job to handle this in an efficient way.

There are more tests and proof-of-concept implementations necessary for all possible solutions. This will be the next step of preparing a reasonable and well-structured architecture for the redesign of EE-Browser.

First simple load tests both for SVG and converted VDCL files (in each case with identical origin) leads to the assumption that the memory usage of fully loaded SVG files (displayed in Internet Explorer or loaded in SVG DOM library based on C#) needs a bit more memory as the opened VDCL file in SimpliCAD. That result could be used as first hint for further discussions.

As base for all following performance and memory usage tests the latest very huge HCV container from Daimler named “A2225405906\_zgs007\_1” were used.

|  |  |
| --- | --- |
| Physical KLB size | 18,2 MB |
| Physical SVG size (Topology) | 101 MB |
| Physical VDCL size (Topology) | 24,6 MB |

|  |  |
| --- | --- |
| Deserialization time KBL file | 2 sec |
| Load time SVG file (Topology) in IE9 | 19 sec |
| Load time VDCL file (Topology) in SimpliCAD | 32 sec |

|  |  |
| --- | --- |
| Memory usage loaded KBL file | 50 MB |
| Memory usage loaded SVG file in IE9 | 1.090 MB |
| Memory usage loaded VDCL file in SimpliCAD | 1.080 MB |

An important aspect during memory usage tests was the different behavior of the system resources for the different processor architecture platforms which can be selected. If the test applications was built on x86 processor architecture the Win7 x64 machine allocated much fewer memory for the application process as it was the case when built on x64 process architecture. The result was that x86 built POC were not able to open very large drawings – an out of memory exception followed. A quite other behavior occurs for POC built in x64 processor architecture: Win7 allows the POC process to allocate a large amount of memory according to real needs of the application.

It looks like that a further redesign implementation should build in AnyCPU processor architecture to be flexible in platform selection and offers the most efficient memory usage on Windows systems.

Further tests and possible improvements regarding the SVG conversion process leads to the result that two major aspects should get attention:

* Simplify graphic for taping information on segment (would reduce memory usage round about 20%)
* Prevent filling connector table cells with white color means rectangle objects with solid hatch style (would reduce memory usage round about 15%)

It should be possible to exchange taped segment graphics with simple rubber lines. The related information comes from the spline curve definition existent in the KBL file. This could be done during conversion process if user selected this option.

The prevention of using filled connector table cells looks a bit more complicated. Therefore it could happened that overlapping (or better underlying) graphical information are existent. This must be checked while creating the cell rectangle. But the final position of the rectangle is not clear to this time (relative coordinates) because the translation will be executed even the complete group will be closed (transformation information on group object). An intelligent handling is needed to may handle this in a proper way if possible at all.

The next step will consists of some discussions to clarify in which direction we should go to define fix settings for creating a very complete specification.