Research Journal of Applied Sciences, Engineering and Technology 11(4): 355-371, 2015
DOI: 10.19026/rjaset.11.1789
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2015 Maxwell Scientific Publication Corp.
Submitted: March 12, 2015 Accepted: March 24, 2015 Published: October 05, 2015

Research Article

An Interactive UML-like Visualization for Large Software Diagrams

[1]Lukas Holy, [3]Ivo Maly, [2, 3]Ladislav Cmolik, [2]Kamil Jezek and [1]Premek Brada
[1]Department of Computer Science and Engineering,
[2]NTIS-New Technologies for the Information Society, European Centre of Excellence, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 8, Pilsen, 30614, Czech Republic
[3]Faculty of Electrical Engineering, Czech Technical University in Prague, Technicka 2, Prague, 16627, Czech Republic

Abstract: As current software keeps growing in size and complexity, the means to visualize its structure become insufficient. Noticeably, standard UML diagrams and their implementations in the industrial tools can depict only diagrams of certain level of complexity. When the complexity rises above this level, the diagrams become no longer visually understandable and start to hinder analytical reasoning. This is mostly a problem of diagrams created during automated reverse-engineering processes. In this study we summarize and validate a new approach for software structure visualization which aims at supporting visual presentation of large software systems. It combines a notation derived from the UML component diagram with tool-supported interaction, utilizing features like hiding of unnecessary information that can be revealed on demand to reduce complexity of the diagrams. To validate the approach, we implemented an experimental tool that provides both the notation and interactive features. The main contribution of this study is an evaluation of the approach through a user study. The results of the user evaluation suggest that the proposed notation in combination with the interactive features allows users to gain insight into a visualized application faster in comparison to standard UML as supported by industrial tools.

Keywords: Complexity, component diagrams, diagram interaction techniques, software visualization, UML, user study

INTRODUCTION

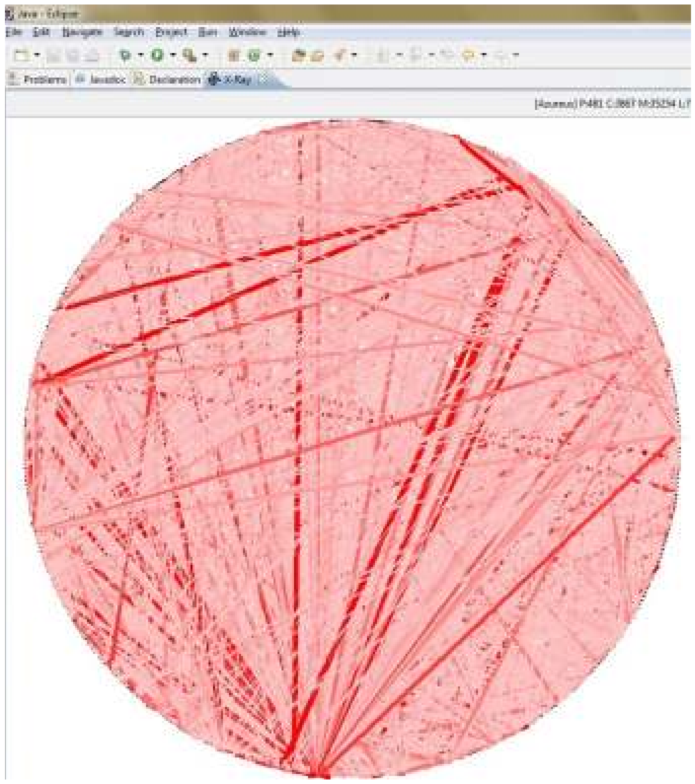
Modern software techniques such as component-based software engineering (Szyperski, 2002) allow for rapid development of large applications. Although components have opened the way to build such applications, designing them is still a difficult task. When a legacy application is to be extended it is also difficult to reverse-engineer it and design its extensions. The main reason is that software applications can easily consist of hundreds or thousands of coarse-grained entities such as modules or components which makes it difficult to explore the structure of their bindings and create a mental model of the whole system.

A user creates a mental model of a reverse-engineered system from the perception, imagination and comprehension of discourse (Johnson-Laird, 1983; Holt, 2002). Although there are lots of tools (Enterprise Architect, 2014; MagicDraw, 2014; IBM Rational Software Architect, 2014; Visual Paragigm, 2014) helping with application design and reverse-engineering, their support for understanding complex applications is still limited. One of the limiting factors is that the tools that build on standard Unified Modelling Language (UML) diagrams cannot cope with large diagrams well. The main problem is the need to provide enough detail and show the overall structure at the same time. Visual clutter is produced by large number of elements and lines. Static nature of UML then cannot hide any of the elements or lines as it is not possible to show it later on demand. Simply said, too many lines of component connections make visualization unreadable.

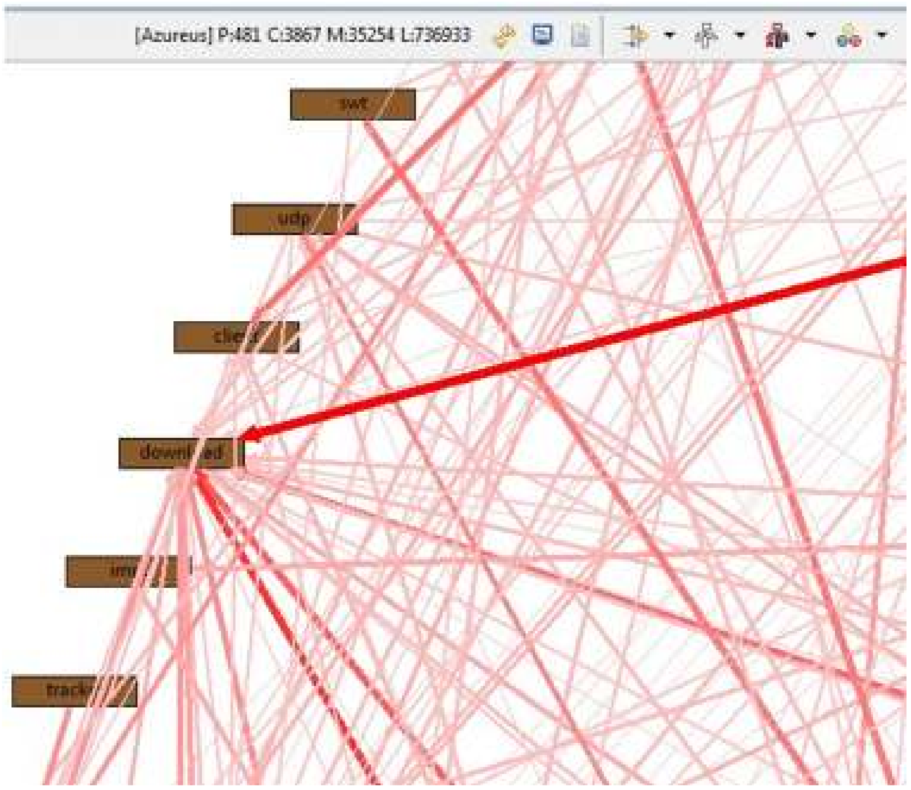
The problem of visual clutter is on the general level illustrated in Fig. 1a. As it may be easily seen from the example, lines denoting component connections hinders orientation in the diagram. On the other hand, when detail is provided (so that, for example, a user is able to read an element name), it is difficult to trace dependencies from distant components. This case is shown in Fig. 1b.

As an alternative, we have proposed a notation derived from the UML component diagram (Caserta and Zendra, 2011) and implemented it together with novel interaction features in a (CoCAEx (Complex Component Applications Explorer) 2014) tool. This tool uses improved representation of the application structure and its non-structural aspects (e.g., metrics) as

Corresponding Author: Lukas Holy, Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia, Univerzitni 8, Pilsen, 30614, Czech Republic



(a)



(b)

Fig. 1: (a): Overview of complex diagram; (b): Higher level of detail in complex diagram

well as the ability to interact with the visualized diagrams. Key parts of this approach were described in our previous works (Holy *et al*., 2012a, b). This new approach should fill the gap mainly in the field of component-based engineering where industrial tools suffer from the above mentioned insufficiency of UML.

In this study we summarise the CoCAEx approach and present its validation study which was performed in order to find out whether it is faster to study the structure of component-based applications interactively rather than statically with one of the state-of-the-art UML tools. The study focuses on performance (time required to provide a correct answer) during application structure analysis tasks.

LITERATURE REVIEW

Software visualization is a well-established field of research. In this section we provide an overview of approaches, techniques and references closely related to our work.

The Unified Modeling Language (UML) provides three groups of diagrams to model both static and dynamic features of software (Object Management Group, 2009), including the component diagram. The notation used in this diagram captures components and their static interconnections. The usability of UML notation has been studied by Moody and van Hillegersberg (2009) and Morris and Spanoudakis (2001).

Research efforts related to visualization of software diagrams fall into two broad categories: displaying the structure and dealing with interactivity. Nowadays, the efforts to display diagrams are most commonly oriented to ex-tensions of the UML itself, not taking interactivity under consideration. However attempts to provide interactivity exist. For instance Dumoulin and Gerard (2010) introduced layers to support multiple views in one diagram and (Byelas *et al*., 2006) suggested the usage of coloured areas of interest to improve orientation in classical UML component diagrams.

The work of Telea and Voinea (2004) on (interactive) visualization of component-based software is generic, but it does not provide many details about components themselves and can hardly be compared with UML. Wettel and Lanza (2007) visualized software as cities. They defined several visual properties of each visualized software element: three dimensions of a block (mapped to defined metrics), position, colour, colour saturation and transparency. This approach could be used for visualization of component application, but it does not provide details needed to gain full comprehension of the application structure.

Interactivity should help primarily with the creation of a mental model, so that one will be able to reason about the architecture and make decisions. It is important to decrease the cognitive load, namely hide unnecessary details as (Holt, 2002) highlighted in several examples. The importance of interactivity for the ability to make decisions about a mental model is mentioned in several studies, e.g., by Meyer *et al*. (2010). They went even further and defined a new science of visually enabled reasoning, implying that interactivity is its key enabler.

Visual clutter reduces understand ability of diagrams as may be seen in Fig. 2. It shows the (Eclipse, 2014) structure visualized in (Dependency Visualization) tool. Some of the techniques to reduce

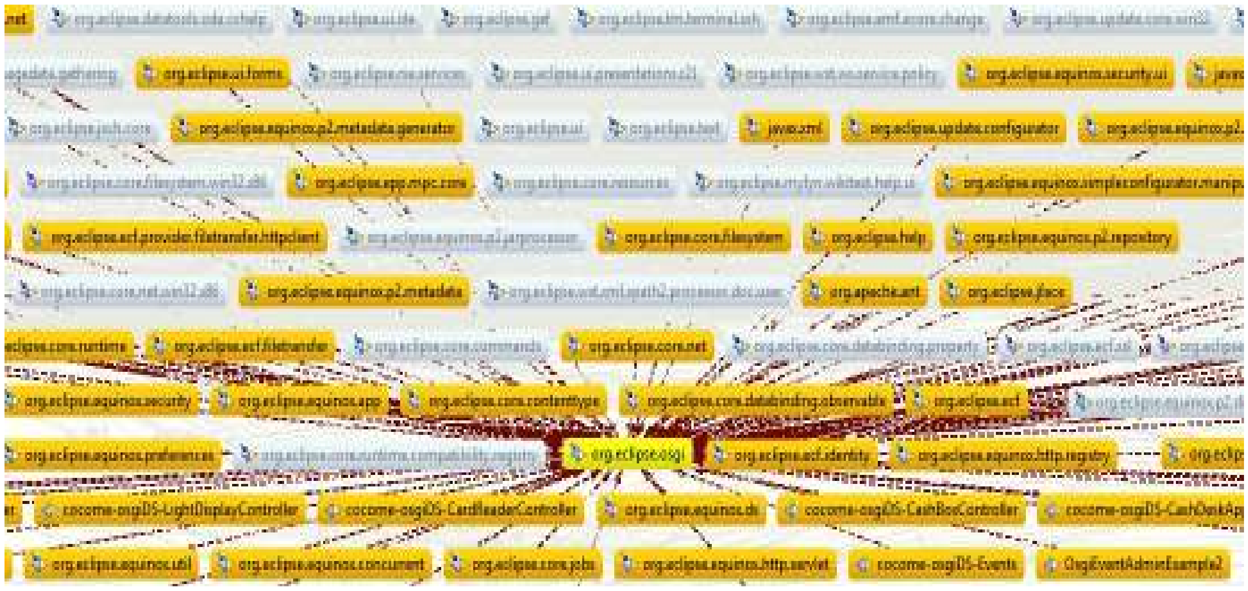

org.eclipse.ui.forms
org.eclipse.equinox.p2.metadata.generator
jwwo.xml
org.eclipse.update.configurator
org.eclipse.app.mpc.core
org.eclipse.equinox.simpleconfigurator.manip
org.eclipse.ecf.provider.filetransfer.httpclient
org.eclipse.core.filesystem
org.eclipse.help
org.eclipse.equinox.p2.repository
org.eclipse.equinox.p2.metadata
org.apache.ant
org.eclipse.jface
org.eclipse.ecf.filetransfer
org.eclipse.core.net
org.eclipse.equinox.security
org.eclipse.equinox.app
org.eclipse.core.contenttype
org.eclipse.core.databinding.observable
org.eclipse.ecf
org.eclipse.equinox.preferences
org.eclipse.osgi
org.eclipse.ecf.identity
org.eclipse.equinox.http.registry
cocome-osgiDS-LightDisplayController
cocome-osgiDS-CardReaderController
org.eclipse.equinox.ds
cocome-osgiDS-CashBoxController
org.eclipse.equinox.util
org.eclipse.equinox.concurrent
org.eclipse.core.jobs
org.eclipse.equinox.http.servlet
cocome-osgiDS-Events
OsgiEventAdminExample2

Fig. 2: Visual clutter caused by highly connected components

clutter were developed to cope with this problem: bundling (Holten and Van Wijk, 2009), sampling (Rafiei, 2005), or clustering (Chen and Liu, 2003). From the well-developed taxonomy of these techniques described by (Ellis and Dix, 2007) the following ones are primarily relevant for us.

Firstly, visual clutter caused by the lines is often reduced by edge bundling (Holten, 2006). Although this approach reduces the clutter, it makes it difficult to trace the dependencies between connected nodes leading through the edge bundles. Secondly, visual clutter can be reduced by the clustering of components (nodes) so that multiple components are visually represented by one node. Thus the number of nodes in the whole diagram is reduced, though the connections among them are usually still present. Clusters can either be marked manually, in an automated way (Chiricota *et al*., 2003; Mancoridis *et al*., 1998) or by a combination of those approaches (McGee and Dingliana, 2012). Thirdly, it is possible to reduce clutter by dividing large diagram into smaller ones. But in this case the user would lose the overview of the whole system and the information about interconnections among system parts. Diagrams of hierarchical component models (Bures *et al*., 2006) usually does not have this problem because they keep the information about parts in their hierarchy but there are lots of component models (Sun Microsystems, 2006; OSGi Alliance, 2009) with flat structure where the described problem can occur. Lastly, the chosen layout algorithm is an influencing factor to clutter since it can ease (or hinder) orientation in both clustered graphs (Feng, 1997) and the non-clustered ones (Hachul and Jünger, 2007).

Existing tools related to the domain of component diagram visualization were described in our previous work (Holy *et al*., 2012c). One of the relevant tools is SoftVision (Softvision, 2014; Telea and Voinea, 2004), a software visualization framework which is able to interactively explore relations between data structures. It is a desktop application which offers a generic interface for describing the language to be parsed as an input. It thus requires initial investment in such description creation. Its visual syntax is not well known.

Works evaluating visualization approaches are also related. The work of Forsell (2010) should be highlighted as it provides a clear guide for similar studies. Laidlaw *et al*. (2001) use a similar comparative study of performance on 2D vector field visualization methods. Evaluation of software visualization was also described by Sensalire *et al*. (2009).

METHODOLOGY

The CoCAEx notation and approach: To visualize a diagram efficiently we need to satisfy several and often contradictory criteria. E.g., on one hand we need to reduce the visual clutter mainly caused by the large number of lines in the diagram, but on the other hand we need to show as much information about interconnections between the components as possible.

We are approaching this problem by utilizing information hiding to reduce the visual clutter and interactivity to display the hidden information on demand. Displaying the hidden information on demand may slow down the user during diagram exploration. However, we believe that the speed up gained by reduced clutter will be much higher.

Based on above desiderata we designed a suitable set of techniques which deal with them; see earlier works (Holy *et al*., 2012b, c) for details. Subsequently we put them into one integrated approach. Because we expect that the most useful features are the ones reducing visual clutter, the presentation here focuses on them-interface clustering, most connected components removal and forming logical clusters. The drawback of these methods is that details are hidden by default and shown on demand which may potentially slow down users during diagram exploration. However, we believe that the speed up gained by reduced clutter will be much higher; the description provided in next subsections and the user study we performed aim to provide support for this position.

The tool which implements the integrated approach, called CoCAEx, is designed for easy and quick exploration of component diagrams. The tool provides standard features such as panning and zooming, but also several more advanced interactive techniques such as exploration of details, full-text search, highlighting and Separated Components area technique (SeCo).

The following subsections describe the key features of the tool. Please refer to Fig. 3 where each feature is marked with a capital letter in a circle and respectively linked in the following text.

Interactive highlighting: Within the diagram area, interactive highlighting is provided to help users quickly identify components related to the selected one. If the user clicks on a component (H), its required connections are highlighted by red colour and the provided ones by blue colour. All other components and connections become visually less prominent. This feature helps the user orientate themselves in diagrams of highly interconnected applications and leads to perceived diagram simplification (Fig. 4 and 5).

Another part of highlighting is a full-text search of components' names. It is provided, together with component highlighting of the results, directly in the visualized diagram.

Interface clustering: To reduce visual clutter, our approach introduces modifications to the UML component diagram. We tried to propose a modification that resembles the original UML notation as much as possible to allow for easy switching from existing tools to the new one.

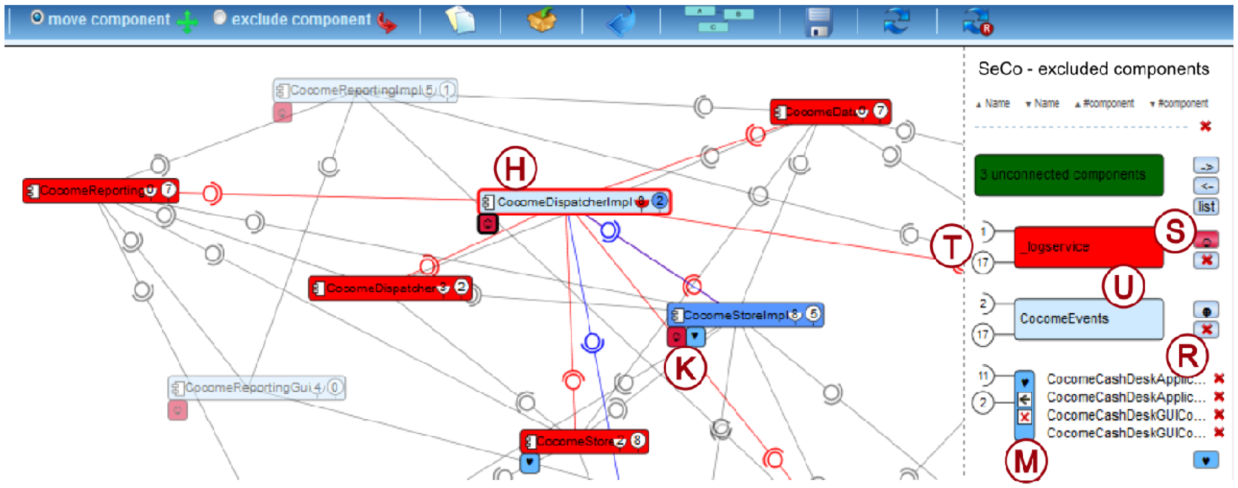


Fig. 3: CoCAEx demonstration

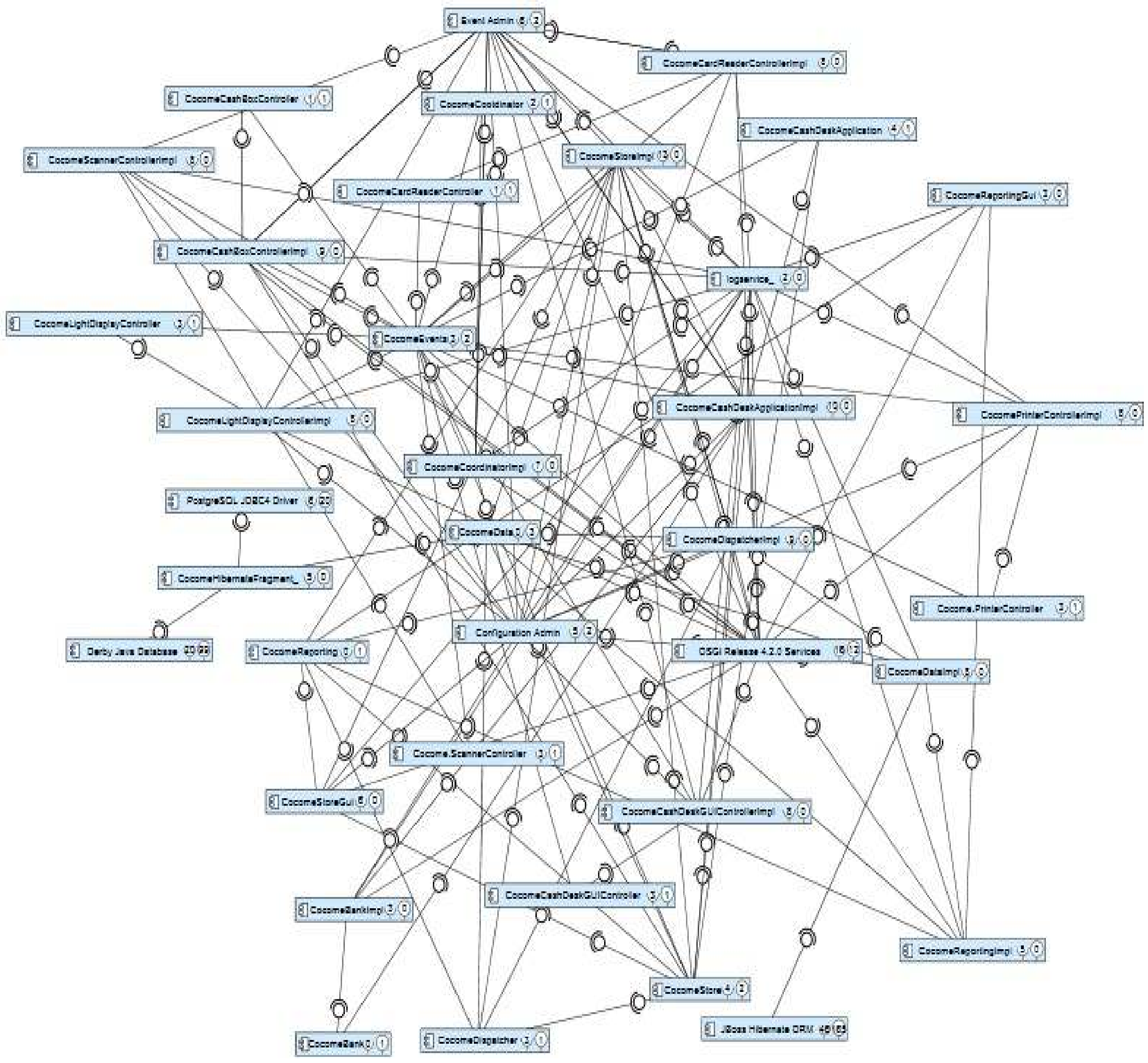


Fig. 4: CoCoME application shown in CoCAEx

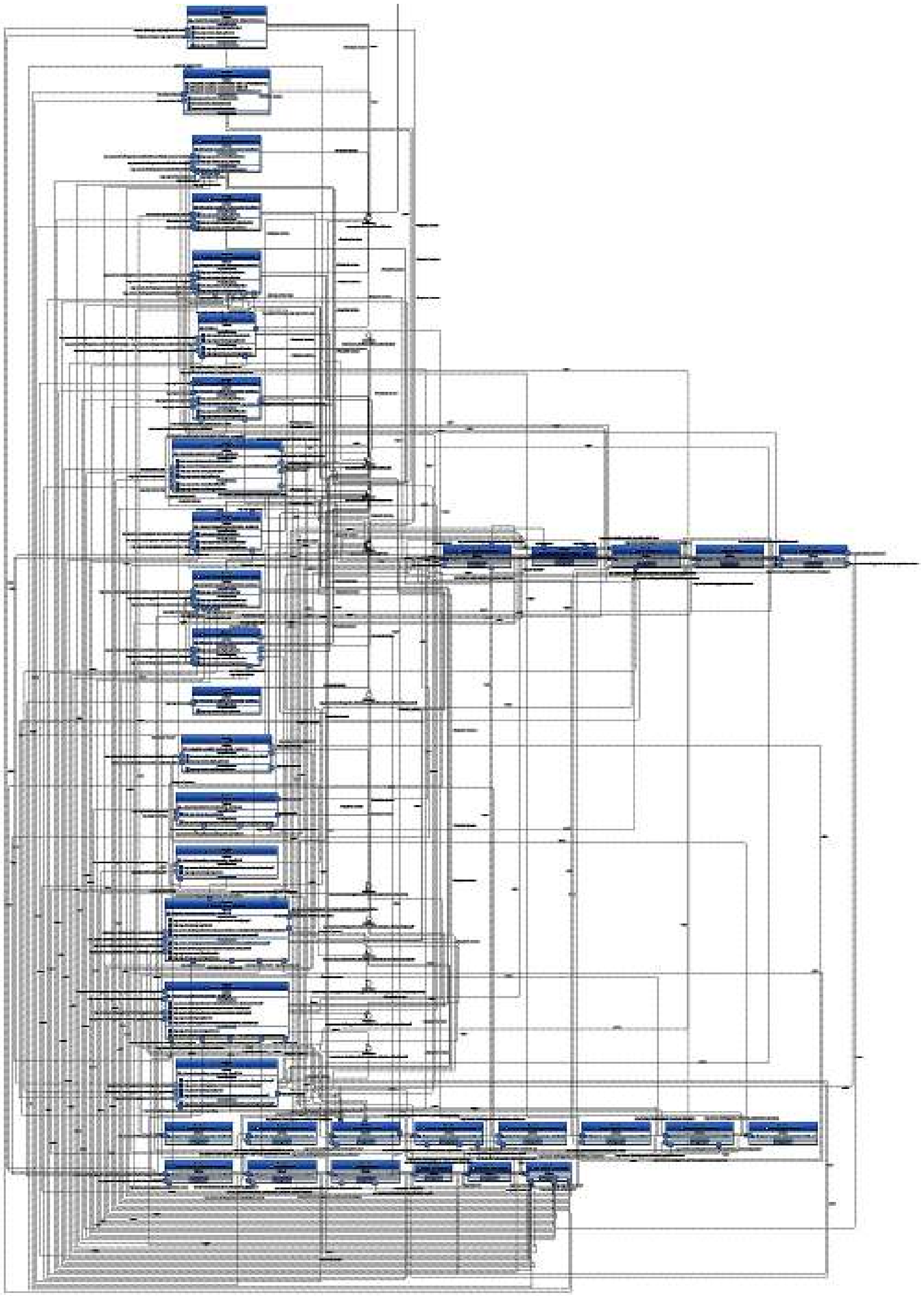


Fig. 5: UML diagram showing the CoCoME application

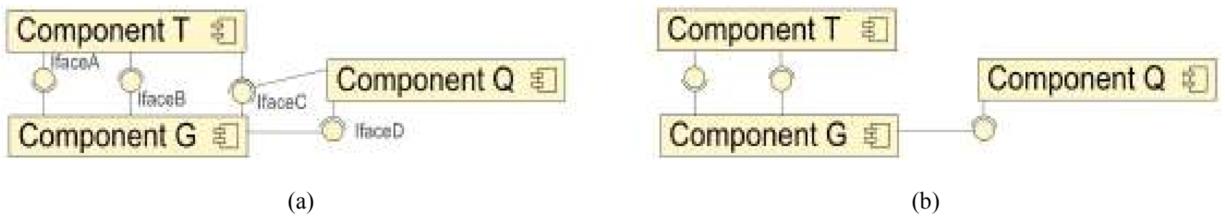


Fig. 6: (a): Differences between UML; (b): Our approach

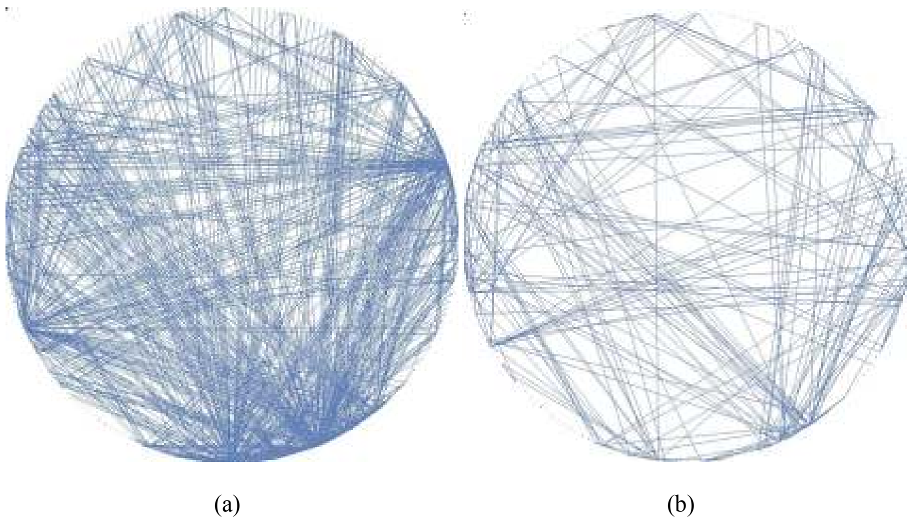
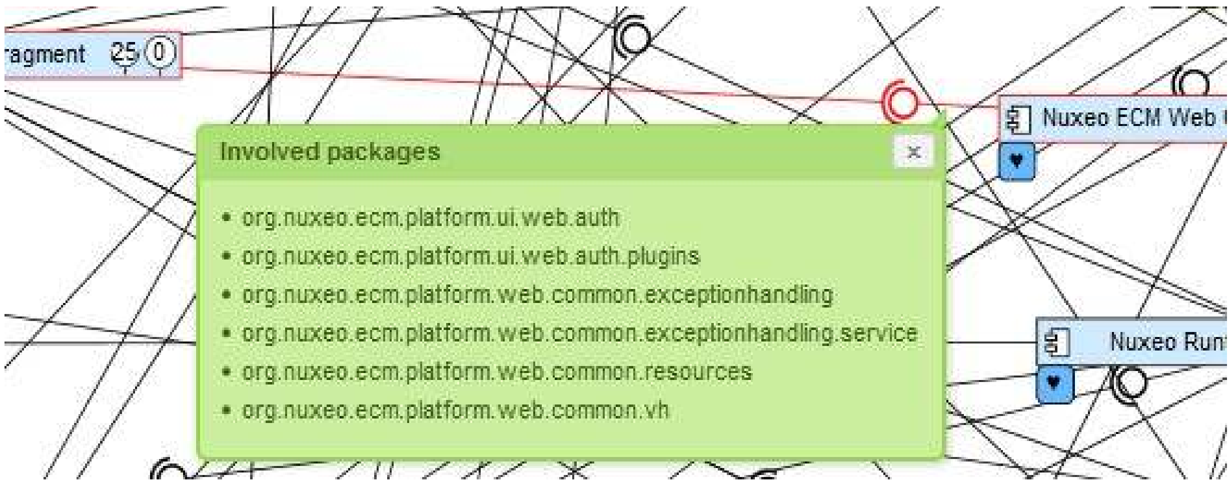


Fig. 7: (a): Visual clutter of Nuxeo-all connections shown; (b): Visual clutter of Nuxeo-7 most connected components removed



- org.nuxeo.ecm.platform.ui.web.auth
- org.nuxeo.ecm.platform.ui.web.auth.plugins
- org.nuxeo.ecm.platform.web.common.exceptionhandling
- org.nuxeo.ecm.platform.web.common.exceptionhandling.service
- org.nuxeo.ecm.platform.web.common.resources
- org.nuxeo.ecm.platform.web.common.vh

Fig. 8: Clustered interfaces exploration

The most important change is interface clustering. In UML, each single interface provided by a component is visualized as a "lollipop shape" and other components that require this interface are connected to this "lollipop shape" with a "socket shape". Our approach on the contrary groups all the respective interfaces of one component into one symbol (i.e., only one "lollipop" and one "socket" are drawn). The consequence is that for each pair of dependent components, only one connection is needed to display the relationship between all the respective provided and required interfaces. This feature is visualized in Fig. 6. In this example, there are two required interfaces (Iface C and Iface D) for Component Q that is provided by Component G displayed as separated "lollipops". Instead, our approach shows only one connection (one "lollipop" and one "socket"). It groups Iface C and Iface D to one graphical element. Names of the interfaces are not shown and can be displayed interactively on demand. When there is only one connection, e.g., Iface A between Component G and Component T, our approach and UML provides the same visual notation, except our notation hides interface names (Fig. 7).

A clustered interface, as mentioned above, can be explored simply by mouse click. CoCAEx then shows the interfaces involved in the connection, as shown in the green tooltip in Fig. 8.

The interface clustering effect on clutter reduction is visually apparent in Fig. 4 and quantified in Table 1. It shows several software systems with components having large numbers of connections. The table lists one system per a line with columns denoting its number of components, clustered and non-clustered connections among the components and the clustering effect that indicates the connections reduction ratio between clustered and non-clustered option. While non-clustered connections represent e.g., the UML notation which connects separately each individual provided-required interface pair, clustered connections collapse all connections between two components into two sets: all provided interfaces and all required interfaces.

Separated components area: We performed several experiments focused on removing the most connected components from a diagram performed on systems listed in Table 1. These experiments tried to examine the correlation between the number of removed components from the graph and the number of consequently removed lines. In one of the experiments only 7 (Nuxeo, 2014) components have been removed from the diagram area leading to 241 and 431 lines remaining in the graph for the clustered and non-clustered versions, respectively. Therefore, the graphs were reduced by 69% of lines in the clustered and 65% of lines in non-clustered version. Visual effect of the results is shown in Fig. 7a and b, using circle layout for clarity. These numbers show that by removing the most connected components, significant visual clutter reduction may be achieved.

Based on these experiments and in line with the aim to keep the information about component interconnections available even when such highly connected nodes have been removed from the diagram, we developed a new technique called the Separated Components area (SeCo). The SeCo technique reduces visual clutter by removing components (e.g., those with a large number of connections) from the main diagram into a separated area placed on the side of the tool's window (a right sidebar in Fig. 3). When a user moves components from the main diagram to this area, the lines between these components and the remaining ones are elided.

Several other features are connected with SeCo. The area is structured into a list of items. Each item consists of clustered connections (indicated by the mark (T) in Fig. 3), components (U) and one corresponding Symbol (S). Connections are clustered into two sets

Table 1: Several systems with the number of components and connections

System	Components	Clustered	Non clustered	Clustering effect (%)
Nuxeo (2014)	202	698	1425	48
CoCoME (2014)	37	125	188	66
OpenWMS (2014)	65	232	642	36
Eclipse (2014)	378	533	1079	49

(T): all provided connections and all required connections (displayed as "lollipops" and "sockets" to resemble standard UML). Numbers inside the clustered connections represent the number of elements clustered in the given symbol. It is a design which goes beyond UML capabilities.

The purpose of the graphical symbols-icons is-to create clear and easily recognizable keys which uniquely identify each of the items within the SeCo area. These symbols can be used in the diagram area to represent connection between a given component and the corresponding item placed in SeCo area (S). They are shown as small rectangles (which we call "delegates") neighbouring the displayed components (K).

Typically, users move components to SeCo area when they want to mark them as "familiar ones" and concentrate on the rest of the system. For this reason, it makes sense to remove all connecting lines to reduce visual clutter and to enable easy exploration of other system parts. Also very often, only a small number of components are connected to a large number of other components. Such components are often, among developers, informally called "God Objects" (Fig. 2). This is another kind of the best candidates for removal to SeCo, because such action significantly reduces the number of lines from the graph. For instance, a user may displace a component implementing a logger. Such a component is probably used by most of components in the system and its displacement reduces the diagram complexity. It is also usually not necessary to show such components in a diagram, because it is actually not important for a user.

Forming logical clusters: It is possible that a particular functionality of the system is implemented by a group of components. Clarity of the diagram is therefore enhanced if such groups are removed from a diagram and replaced by a simpler element. The SeCo part of our approach therefore provides the concept of "groups" to handle such situations, indicated by the Mark (M) in Fig. 3.

Any component from the diagram area can be added to an existing item (a component or a group) in SeCo. This is achievable by the right-click action on a component in the diagram area, as shown in Fig. 9. A context menu appears and a user can select the symbol which belongs to the item in SeCo. If a user is not sure what an item with corresponding symbol contains, there is a possibility to expand such an item directly in the context menu as shown for the red group.

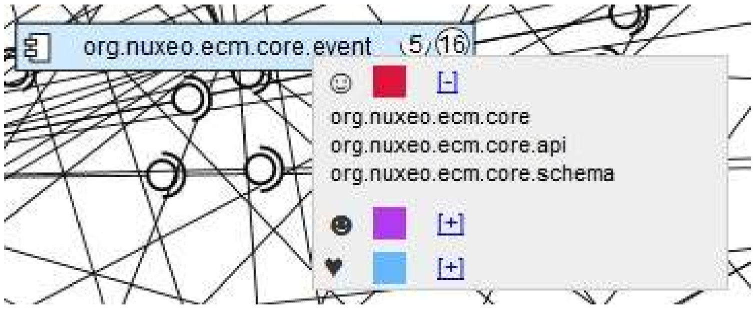


Fig. 9: Adding components from diagram to SeCo groups

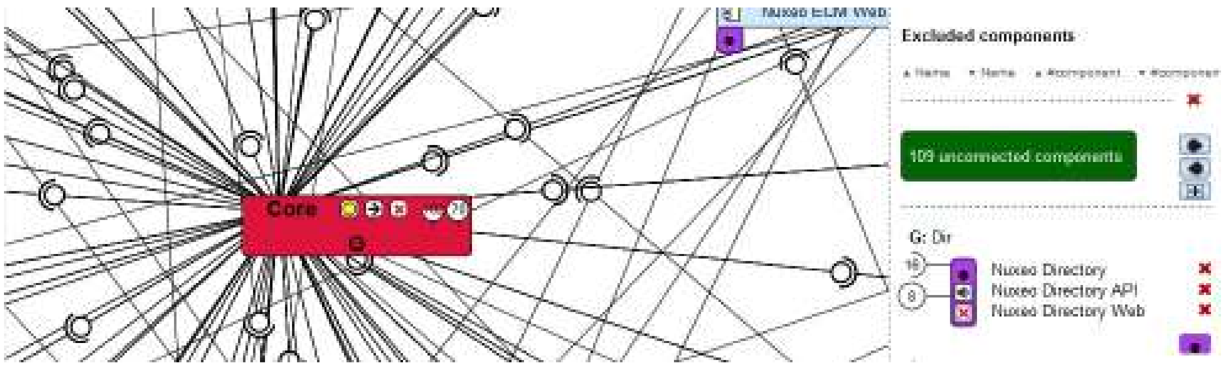
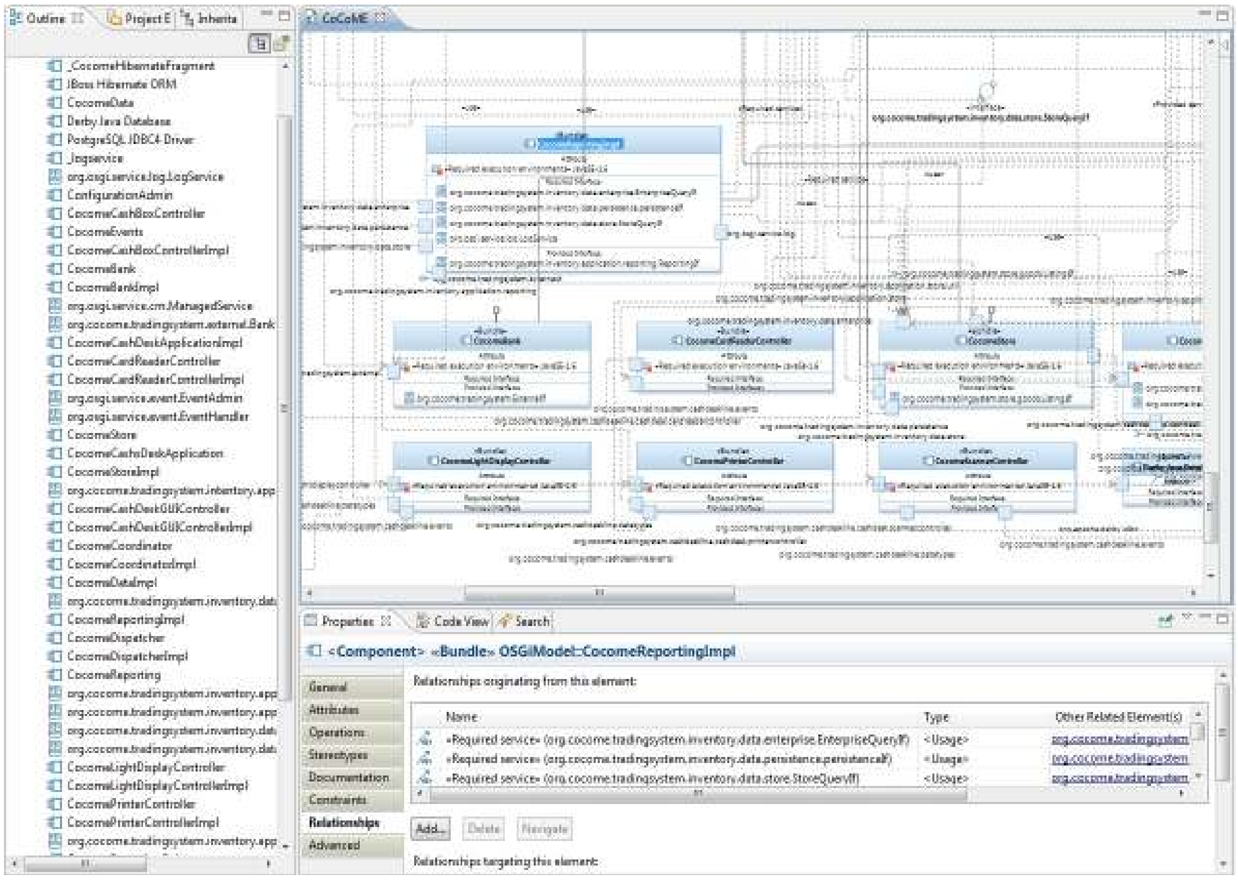


Fig. 10: Showing the "core" group as a node



_CocomeHibernateFragment
JBoss Hibernate ORM
CocomeData
Derby Java Database
PostgreSQL JDBC4 Driver
_logservice
org.osgi.service.log.LogService
ConfigurationAdmin
CocomeCashBoxController
CocomeEvents
CocomeCashBoxControllerImpl
CocomeBank
CocomeBankImpl
org.osgi.service.cm.ManagedService
org.cocome.tradingsystem.external.Bank
CocomeCashDeskApplicationImpl
CocomeCardReaderController
CocomeCardReaderControllerImpl
org.osgi.service.event.EventAdmin
org.osgi.service.event.EventHandler
CocomeStore
CocomeCashDeskApplication
CocomeStoreImpl
org.cocome.tradingsystem.inventory.app
CocomeCashDeskGUIController
CocomeCashDeskGUIControllerImpl
CocomeCoordinator
CocomeCoordinatorImpl
CocomeDataImpl
org.cocome.tradingsystem.inventory.dat
CocomeReportingImpl
CocomeDispatcher
CocomeDispatcherImpl
CocomeReporting
org.cocome.tradingsystem.inventory.app
org.cocome.tradingsystem.inventory.app
org.cocome.tradingsystem.inventory.dat
org.cocome.tradingsystem.inventory.dat
CocomeLightDisplayController
CocomeLightDisplayControllerImpl
CocomePrinterController
CocomePrinterControllerImpl
org.cocome.tradingsystem.inventory.app

<Component> «Bundle» OSGiModel::CocomeReportingImpl

General
Attributes
Operations
Stereotypes
Documentation
Constraints
Relationships
Advanced

Relationships originating from this element:

Name	Type	Other Related Element(s)
«Required service» (org.cocome.tradingsystem.inventory.data.enterprise.EnterpriseQueryIf)	<Usage>	org.cocome.tradingsystem
«Required service» (org.cocome.tradingsystem.inventory.data.persistence.persistanceIf)	<Usage>	org.cocome.tradingsystem
«Required service» (org.cocome.tradingsystem.inventory.data.store.StoreQueryIf)	<Usage>	org.cocome.tradingsystem

Add... Delete Navigate

Relationships targeting this element:

Fig. 11: An outline and properties view of IBM rational software architect

The group in SeCo contains a list of components and is represented by one symbol, as shown by the heart symbol near the Mark (M) in Fig. 3. Logical clusters can be formed and displaced from the diagram to reduce its visual complexity. When the cluster is displaced, a user can continue with studying the remaining diagram. All connections to any of the components in such group use a common delegate in the diagram. Purpose of sharing the delegate is again the desire for space saving.

Besides the representing symbol, any group can also have a name. This helps giving the clusters human-understandable semantics better than using only symbols that are primarily for showing elided connection lines.

When a logical cluster of components is formed, there is a possibility to show the whole group which it represents as a diagram node in the diagram area. It is shown in Fig. 10. This allows a user to see connections of this group with the rest of the content in the diagram area directly without any additional interaction.

Compared to single component visualization, we can see that there are additional icons in the symbol's visual representation in the diagram area.

The first one serves for expanding the group's graphical representation from a symbol to a list of components. The list is shown in the diagram area inside a group box. Individual components inside the expanded list are equally interactive as when shown separately. It means a user can use highlighting of its required and provided interfaces. The second one (right arrow) moves the group back to SeCo. The third one (red cross icon) removes the group and shows the individual contained components in the diagram area. The group shown in the diagram area can also use highlighting of required and provided components.

The described approach of forming logical clusters allows users to persistently enrich a diagram by its own point of view on how components should be grouped into individual features or parts. It is suitable for clusters of components a user is interested in and thus wants to keep them together to easily see their surroundings. Alternatively, this feature can be used for components a user is not interested in and wants to keep them together because of their hiding and consequential clutter reduction. When a user finishes diagram exploration and creates the desired clusters, it is then possible to show them in the diagram area and thus provide a high level overview of the system on a higher level of detail than for individual components. Also the visual clutter caused by connections would likely be highly reduced.

Implementation: CoCAEx, 2014 is a web-based tool able to automatically reverse engineer a whole component-based application implemented in a supported component model. It uses the ComAV (Snajberk *et al*., 2012) platform to parse the distribution packages of existing components of an application into a graph based representation. After the ComAV platform creates the model of the application, CoCAEx shows the application diagram in the web page. The front end of CoCAEx uses HTML5 technologies to provide modern and easily extensible user interface.

The UML approach: As has been said, our approach tries to use good parts of UML and propose improved notation implemented in an interactive tool. This decision has been taken as UML is an industry acceptable standard and it is widely used.

A lot of tools implement UML with certain set of interactive features. Typical ones are scrolling, panning, zooming, search or filtering. While it makes the navigation better than with the diagram printed on the paper, it still relies on the statical nature of a diagram keeping its original pros and cons. Therefore, one of the state-of-the-art UML tools should be selected to objectively investigate its usability; this is not an easy task as there are a lot of tools with large (and not fully comparable) sets of features.

For the evaluation of our approach, from the currently most used tools (mentioned in above section) we chose IBM Rational Software Architect (2014) (RSA). The reason for selecting RSA is that it can be considered as the most advanced, industry-strength tool with a lot of additional features and widely used by the biggest software houses. In other words, we decided to choose the most challenging competitor to compare with CoCAEx.

Besides all standard features included in other tools, RSA supports some advanced ones that allow users to manipulate the diagram, like changing the layout of nodes, changing the line routing and modifying the look of components and interfaces. Another added value is in its "properties view" displayed at the bottom of the screen; Fig. 11. This view shows all the details about components and relations and, most importantly, can be used to navigate to related components. For example, the "Relationships" tab shows a list of all elements that use or are used by a given component. This list clearly specifies which kind of relation is used and which components are related. Each relation line contains a link which can be used for acquiring more information about the line.

User study: In order to find out to what degree of interactivity is useful or which techniques used in CoCAEx approach are most beneficial for shortening the time needed for exploring the structure of complex component-based applications, we performed a controlled user study. This section provides the details of its goal and mechanics, while the following sections describe and discuss the results.

Goal of the study: This study evaluates whether it is faster to analyse the structure of component-based applications interactively with the CoCAEx notation and tool rather than to study the structure with one of the good state-of-the-art UML tool, in particular RSA. Our null hypothesis is that the user performance is approximately the same. This knowledge of the effects of the approach is important because the level of interactivity used in CoCAEx is high and could negatively affect the user's performance while he/she collects multiple detailed information, specifically because a lot of this information is hidden and revealing it requires user interaction.

In the study we are using two different tools for experiments. Ideally, our notation as well as standard UML notation would be tested in one tool to use exactly the same look and feel for both notations as well as the same basic framework of user interaction. However, this was not possible as CoCAEx cannot be switched to standard UML notation and our notation is obviously not implemented in RSA. Integration of one of the notations to the opposite tool is technically difficult. For that reason, we decided to perform the study with two tools and mitigate the possible problems caused by their differences by a careful design of the experiment procedure.

The study simulated the activities performed during one step of architecture analysis. These activities are focused on collecting knowledge about components' features, dependencies and overall context consisting of related components. The concrete set of tasks used in the study, which was based on those activities, is discussed thoroughly further below.

Participants: Twelve participants were recruited from two different universities. All participants were young software engineers and all were academics or Ph.D. candidates (the use of academics and Ph.D. candidates was encouraged by Sensalire *et al*. (2009). All were proficient users of computers and had sufficient knowledge of UML to fully understand the presented diagrams and they had also proficiency in analysis of component-based structures and applications.

Apparatus: The hardware used in the study consisted of standard PC (Intel Core i5 at 2.8 GHz with 8 GB RAM), 24 inches LCD display (resolution 1920×1080 pixels), PC keyboard and optical mouse with 2 buttons, running Windows 7.

Participants used RSA (version 8) and CoCAEx Tool (2014) (version 0.3) in the study. Although RSA is currently at version 9, there are no new features introduced in eclipse client, which would affect results of our study. Both tools were running smoothly on the selected hardware.

During the study, users were analysing the Common Component Modeling Example application CoCoME (CoCoME, 2014; Rausch *et al*., 2008) -an information system for supermarket chains developed originally with the aim of comparing different approaches to component-based software modelling. The CoCoME application, which represents a medium-size application, consists of about 40 components divided into three main parts. First is a cash desk, including barcode scanners, credit card readers, etc. The second part is a store back office server and a store client. Finally, the chain part consists of an enterprise server and client applications. For the purposes of the test we have used our own implementation of CoCoME (available at CoCoME implementation in OSGi (2014)) in the OSGi component model (OSGi Alliance, 2009) - the diagrams in Fig. 4 and 5 visualize the structure of this CoCoME implementation in CoCAEx and UML, respectively.

The UML diagram (in RSA) of the CoCoME application can be seen in Fig. 5. There are components containing names, additional details and large number of non-clustered connections. If we compare this Figure to CoCoME shown in CoCAEx application (Fig. 4), we can see that interface clustering used in CoCAEx reduced the number of connections. Also details of components hiding saves space in the diagram area.

Design: The study was organized as one factor (with two levels) within-subject design. The independent variable was the used analytical tool. The order of tools was counter balanced and the group effect (asymmetrical transfer of skills from tool one to tool two) was evaluated.

The main measure was speed, measured as a number of seconds needed to accomplish each task.

Procedure: The test was performed at two locations with the same procedure. At each location, the experiment was performed in a dedicated room where participants were not disturbed. Before the experiment was started, participants adjusted the position of the display and the mouse to feel comfortable.

The moderator of the experiment first explained the user interface of the first tool. The experiment began with a training session. In the training session participants were asked to accomplish 5 tasks very similar to those they will perform in the actual experiment, but using visualized structure of different small-size application. During the training session, the moderator helped participants to accomplish all tasks if necessary. The goal was to let participants get familiar with the first approach, get used to the experiment procedure and minimize any learning effects. Specifically, we focused on minimization of impact of different complexity of tools on the results of the study. Namely, in case of RSA, we gave hints to users of what features they will need in the study. This should prevent a situation when the user gets lost in large number of features provided by RSA. The training lasted 15-20 min and it ended when the participant felt confident and familiar with both tools and able to perform all types of tasks used in this user study.

Training was followed by the experiment session with the first tool. During the experiment session, participants were asked to proceed as quickly as possible. All tasks were performed so that the right result was obtained in the end. Between each task, participants were allowed to take a short break. After the sessions with the first tool the same procedure was repeated for the second tool. The whole study lasted about 1 h.

Participants accomplished 5 tasks. These were given to participants as follows:

T1 : **Which components use interfaces provided by CoCoME data-Impl?** The task was focused on analysis of the parts of the system which will be influenced if some particular component is changed.

T2 : **Which components are not from CoCoME core (are third party)?** The task was focused on analysis of the system structure, mainly discovery of the core of the system and usage of the third party components in the system.

T3 : **Which packages need CoCoME data Impl from CoCoME data?** The task was focused on analysis of the relation between two components in the system.

T4 : **Which components do not require or provide interfaces to any other components (are unconnected)?** The task was focused on analysis of unconnected components that are suspicious, because they are probably using some non-standard way of communication with other components.

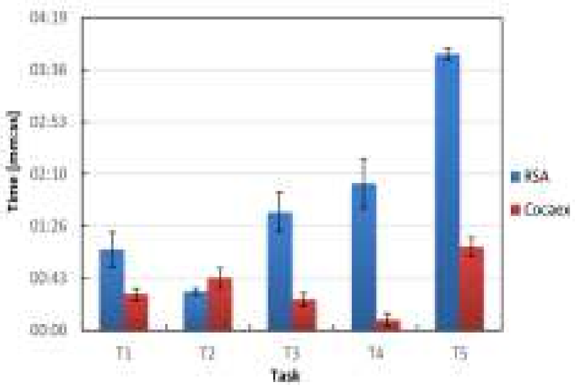
T5 : **Which components require or provide interfaces to any of cash desk components in CoCoME?** The task was focused on analysis how a particular part (usually feature) of the system is connected with the rest of the system.

The tasks were defined based on our experience with the structure of component-based applications and based on hints obtained during interviews with several software engineers from local software companies. Individual tasks are typical tasks used iteratively in global task that deals with the question of what is the structure of application and how particular components are integrated in the CoCoME application. One has to find out what these components offer and require and uncover their ties to other components, simulating the activities performed during one step of the architecture analysis.

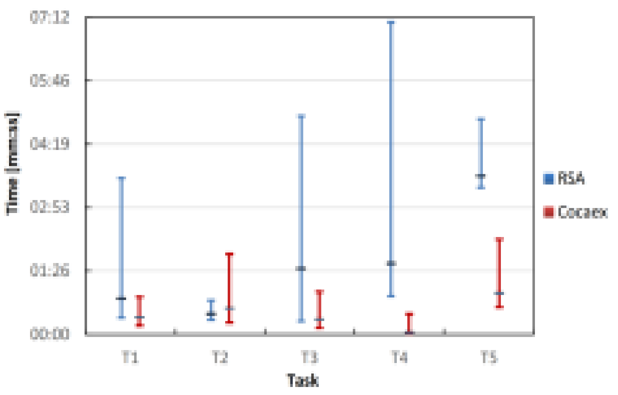
Two questionnaires were given to participants during the test. At the beginning of the test, participants were asked about the experience with UML diagrams and UML editors. After the data collection, participants completed a questionnaire investigating their subjective judgment about the used approaches.

RESULTS AND DISCUSSION

This section provides detailed results of the study for each approach and their comparison. As the reader may notice, the results differ greatly depending on the participant. This was caused by individual perception and orientation abilities. A lot of attention was paid to preparing all participants thoroughly in the above section. Task completion times of all participants for each task and basic statistics of individual tasks are presented in Table 2. Comparison of mean times of individual tasks is in Fig. 12a. Detailed comparison of average, median, minimal and maximal times is in Fig. 12b.



(a)



(b)

Fig. 12: (a): Comparison of average times needed to accomplish the tasks in RSA and in CoCAEx; (b): Minimum and maximum times with marked medians (black lines) needed to accomplish the tasks in RSA and in CoCAEx

Table 2: Time (min: sec) measured for each participant and each task. Together with sum of times for all tasks and statistical measures

	RSA						CoCA-Ex					
Participant	T1	T2	T3	T4	T5	All	T1	T2	T3	T4	T5	All
P1	00:33	00:39	01:34	01:07	03:47	07:40	00:23	00:29	01:00	00:30	01:06	03:28
P2	02:03	00:30	01:45	01:38	04:55	10:51	00:26	00:41	00:31	00:12	02:11	04:01
P3	01:05	00:30	01:18	01:47	03:24	08:04	00:23	00:34	00:28	00:05	01:39	03:09
P4	01:09	00:32	00:28	03:02	03:23	08:34	00:16	00:46	00:11	00:05	00:58	02:16
P5	03:35	00:30	04:59	07:06	04:37	20:47	00:41	01:51	00:32	00:11	01:52	05:07
P6	00:54	00:26	00:21	01:02	03:26	06:09	00:25	00:55	00:15	00:05	00:39	02:19
P7	00:26	00:32	00:27	01:53	03:22	06:40	00:46	00:29	00:22	00:10	00:47	02:34
P8	00:50	00:32	01:57	00:55	03:36	07:50	00:22	00:49	00:23	00:05	00:57	02:36
P9	00:45	00:49	02:15	01:43	03:52	09:24	00:38	00:50	00:26	00:05	00:58	02:57
P10	00:57	00:27	01:31	01:17	03:31	07:43	00:14	00:23	00:21	00:06	00:54	01:58
P11	00:40	00:23	01:34	01:41	04:15	08:33	00:53	00:18	00:13	00:04	00:41	02:09
P12	00:27	00:37	01:25	01:11	03:40	07:20	00:30	00:33	00:20	00:05	01:07	02:35
Measure	T1	T2	T3	T4	T5	All	T1	T2	T3	T4	T5	All
Mean	01:07	00:32	01:38	02:02	03:49	09:08	00:30	00:43	00:25	00:09	01:09	02:56
Median	00:52	00:31	01:32	01:39	03:38	07:57	00:26	00:38	00:23	00:05	00:58	02:36
Min.	00:26	00:23	00:21	00:55	03:22	06:09	00:14	00:18	00:11	00:04	00:39	01:58
Max.	03:35	00:49	04:59	07:06	04:55	20:47	00:53	01:51	01:00	00:30	02:11	05:07
S.D.	00:51	00:07	01:10	01:37	00:30	03:42	00:12	00:23	00:12	00:07	00:28	00:52

For statistical analysis of the results we have used the repeated measures ANOVA (Analysis of Variance) (MacKenzie, 2012). We report the results in F-statistics notation $F_{X,N} = V$, $p<\alpha$ where X is number of factors considered in the study, N is number of participants in between-subject design (note that correction is applied in our case due to the within-subject design). If value $V\leq 1$ then it is impossible for the means of the tested approaches to be significantly different. The amount V rising above 1 is an indication of the size of the difference in the means of the tested approaches. If the null hypothesis is true then p is the probability of obtaining the observed data. In other words, if the value of p is lower than confidence level α (α is typically 0.05) then the difference in the means of the tested approaches are unlikely to occur in the view of the null hypothesis.

From this follows that we can reject the null hypothesis when $V>1$ and $p<0.05$. When $V\leq 1$ or $p\geq 0.05$ were port that the difference in means of the tested approaches is not significant (shortened to "ns"), in such case the null hypothesis cannot be rejected.

In the following subsections we evaluate the null hypothesis separately for each task. Later, in the summary, we evaluate the null hypothesis for all tasks together.

Task T1: Which components use interfaces provided by CoCoME data-Impl? There was a significant difference in speed between RSA and CoCAEx ($F_{1,11}$ = 5.758, $p<0.05$). The average speed for RSA was 67 sec and for CoCAEx 29.75 sec (2.25× faster than RSA). The group effect was not detected.

In both cases, the users searched for the component and then they analysed its provided interfaces. For RSA, the analysis of provided interfaces took longer time, while for CoCAEx this information was immediately visible, when the component was selected.

Task T2: Which components are not from CoCoME core (are third party)? There was no significant difference in speed between RSA and CoCAEx ($F_{1,11}$ = 2.327, nsec). The average speed for RSA was 32.25 sec (1.34× faster than CoCAEx) and for CoCAEx 43.17 sec.

This was the only task where RSA was faster than CoCAEx, but not significantly. With CoCAEx, the users were searching for the components in the graph, which took them additional time to skim through the graph. On the other hand, with RSA, the users searched for the components in the list in Outline window, Fig. 11, which was very quick.

Task T3: Which packages need CoCoME data Impl from CoCoME data? There was a significant difference in speed between RSA and the CoCAEx ($F_{1,11}$ = 12.954, $p<0.05$). The average speed for RSA was 97.83 sec and for CoCAEx 25.17 sec (3.89× faster than RSA). The group effect was not detected.

In this task CoCAEx allowed fast analysis of imported/exported packages between two components, which is not immediately visible in the graph, but it is shown as popup when mouse is over the interconnection between two components. In RSA, participants usually used the project tree and compared full lists of exported/imported packages.

Task T4: Which components do not require or provide interfaces to any other components (are unconnected)? There was a significant difference in speed between RSA and CoCAEx ($F_{1,11}$ = 14.905, $p<0.05$). The average speed for RSA was 121.83 sec and for CoCAEx 8.58 sec (14.20× faster than RSA). The group effect was not detected.

In this task the difference in performance between RSA and CoCAEx was the biggest. This was due to the fact that with RSA the participants had to find out the unconnected components manually, however CoCAEx allowed semiautomatic extraction and listing of all unconnected components in the panel on the right.

Task T5: Which components require or provide interfaces to any of cash desk components in CoCoME? There was a significant difference in speed between RSA and CoCAEx ($F_{1,11}$ = 516.403, $p<0.05$). The average speed for RSA was 229 sec and for CoCAEx 69.08 sec (3.31× faster than RSA). The group effect was not detected.

This was the most complex task which focused on searching of multiple CashDesk components and analysis of interface connections between these components and rest of the components in the CoCoME application. With RSA, this task required searching of CashDesk components in the graph and iteration through their provided/required inter-faces. With CoCAEx, there were two approaches how to achieve this task, both of them took similar time. In the first case, the participants analysed CashDesk components' interfaces in the graph, similarly as in the Task 1. Alternatively, participants extracted CashDesk components to the SeCo area (right sidebar) into one group or individually and then highlighted all components connected through required/provided interfaces.

Subjective evaluation: We also asked participants about the orientation in the visualized structure of the CoCoME application, the level of comfort while working with the tools and any other suggestions.

All participants described the orientation in structure of the CoCoME application visualized in our notation as better. Further, all participants stated that it was more comfortable to solve all tasks in CoCAEx than in RSA. Four participants stated that the visual highlighting of the components related to the selected component significantly improved their orientation in the structure of the application. Three participants suggested allowing the selection of more than one component and visually highlighting components related to all selected components. Two participants would like to use the same approach as used in CoCAEx for components also for classes and class diagrams.

Discussion and summary: The measured times needed to accomplish each task by each participant demonstrate that the tasks T1, T3, T4 and T5 were accomplished on average significantly faster in the CoCAEx Tool (2014) than in RSA. The only exception is the task T2, which the participants accomplished on average faster with RSA. Overall speed comparison showed that the tasks with the CoCAEx Tool (2014) were accomplished on average 3× faster than with RSA; Fig. 12a. In the majority of tasks, the maximum values in CoCAEx were lower than the median values for RSA; Fig. 12b. These results show that even the slower CoCAEx users were faster than at least half of the users in RSA for majority of tasks. The median values are averagely placed at first fifth of the minimum-maximum range.

The results indicate that the structure of the CoCoME application visualized in our notation is less cluttered which leads to faster orientation in the structure of the application. Further, the visual highlighting and interaction features of components related to the selected component provided by CoCAEx allowed participants to visually detect the related components much faster.

From these facts, we can conclude that the abilities to interact with the visualized structure of component-based software systems and to provide visual cues to ease identification of related components is better with CoCAEx. There was a significant difference in speed between RSA and CoCAEx for summed accomplished task times ($F_{1,11}$ = 46.581, $p<0.05$). Results for individual and summed accomplished task times allows us to reject the null hypothesis, i.e., that both tools provide the same speed of analysis of component-based application.

From the subjective point of view the participants perceived the orientation in our notation as better in comparison to the standard UML notation. They also subjectively perceived comfort of work in CoCAEx as better in comparison to RSA. All participants also answered that CoCAEx provides a clearer diagram that is more readable and understand-able. They mentioned two main reasons: highlighting and simplicity of use.

There is a combination of two factors influencing participants' performance in each task, interactivity of tools and visual notation. This fact means that we cannot exactly separate impact of interactivity and the notation. As a consequence, we cannot be sure whether better performance in certain tasks was caused more by the interactivity or by the notation. For instance, we could think that a task was performed faster due to an interactive feature (e.g., usage of interface clustering) while in fact a better colour encoding or contrast helped a user more.

CONCLUSION AND RECOMMENDATIONS

This study presented key parts of our approach for component application visualization and described a user study that compared its performance in component diagram analysis with the UML component diagram. Our notation was implemented and tested in the research proof of the concept CoCAEx Tool (2014), while IBM Rational Software Architect (2014) was selected to test standard UML and certain interactivity features. This software was chosen also because it is widely adapted by industry and largely used by the world biggest software houses.

In the user study, 12 participants were tested in a within-subject test on both tools. The participants accomplished 5 tasks, related to the activities needed in software architecture analysis and their speed to accomplish each task with each tool was measured. In addition, we collected subjective evaluation of the tool usability.

The data obtained shows that participants working with CoCAEx Tool (2014) and its improved notation are comparatively fast of faster (up to 3 times) than participants using standard UML in RSA.

We believe that results of this user study confirm that the proposed interactive visualization provides better user experience, leading to more efficient exploration of complex component-based applications. Features such as highlighting or grouping help users perform faster the exploration while the increased need for interaction does not introduce significant slowdown.

The present implementation of the CoCAEx approach allowed us to perform the validation experiments presented in this study. In our future work, we will also consider implementing the CoCAEx as (Plugin, 2014) to a known tool, such as IBM RSA. This would allow us to compare the results without being affected by differences between tools.

We believe that some of the presented ideas can be generalized to be used in other domains (e.g., UML Class diagrams, bank transfers graphs, server interconnections graphs) where one suffers from visual clutter caused by the large number of nodes and connection lines. Thus one direction of our future work will be providing a generalized concept of the ideas and techniques presented in this study.

ACKNOWLEDGMENT

The study was supported by European Regional Development Fund (ERDF), project "NTIS-New Technologies for the Information Society", European Centre of Excellence, CZ.1.05/1.1.00/02.0090, by the UWB grant SGS-2013-029 Advanced Computer and Information Systems and by the Aktion OE/CZ grant number 68p5.

We would like to also thank Jaroslav Snajberk and Eduard Chromik for the help with the tasks and study design.

REFERENCES

Bures, T., P. Hnetynka and F. Plasil, 2006. SOFA 2.0: Balancing advanced features in a hierarchical component model. Proceeding of the 4th International Conference on Software Engineering Research Management and Applications, pp: 40-48.

Byelas, H., E. Bondarev and A. Telea, 2006. Visualization of areas of interest in component-based system architectures. Proceeding of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, pp: 160-169.

Caserta, P. and O. Zendra, 2011. Visualization of the static aspects of software: A survey. IEEE T. Vis. Comput. Gr., 17(7): 913-933.

Chen, K. and L. Liu, 2003. A visual framework invites human into the clustering process. Proceeding of 15th International Conference on Scientific and Statistical Database Management, pp: 97-106.

Chiricota, Y., F. Jourdan and G. Melanc, 2003. Software components capture using graph clustering. Proceeding of the 11th IEEE International Workshop on Program Comprehension (IWPC '03), pp: 217.

CoCAEx Tool, 2014. Retrieved form: http://relisa-dev.kiv.zcu.cz:8083/efpcocaex/. (Accessed on: Sep. 29, 2014)

CoCoME, 2014. Retrieved form: http://www.cocome.org/. (Accessed on: Sep. 29, 2014)

CoCoME Implementation in OSGi, 2014. Retrieved form: http://relisa-dev.kiv.zcu.cz/data/experiments/cocome-userstudy/. (Accessed on: Sep. 29, 2014)

Dumoulin, C. and S. Gerard, 2010. Have multiple views with one single diagram! a layer based approach of UML diagrams. Research Report Inria-00527850, Institut National de Recherche en Informatique et en Automatique, Universite des Sciences et Technologies de Lille (October, 2010).

Eclipse, 2014. Retrieved form: https://www.eclipse.org/. (Accessed on: Sep. 29, 2014)

Ellis, G. and A. Dix, 2007. A taxonomy of clutter reduction for information visualisation. IEEE T. Vis. Comput. Gr., 13(6): 1216-1223.

Enterprise Architect, 2014. Retrieved form: http://www.sparxsystems.com.au/ (Accessed on: Sep. 29, 2014).

Feng, Q., 1997. Algorithms for drawing clustered graphs. Ph.D. Thesis, Department of Computer Science and Software Engineering, University of Newcastle.

Forsell, C., 2010. A guide to scientific evaluation in information visualization. Proceeding of 17th International Conference on Information Visualisation, pp: 162-169.

Hachul, S. and M. Jünger, 2007. Large-graph layout algorithms at work: An experimental study. J. Graph Algorithms Appl., 11(21): 345-369.

Holt, R., 2002. Software architecture as a shared mental model. Proceeding of 1st ASERC Workshop on Software Architecture.

Holten, D., 2006. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE T. Vis. Comput. Gr., 12(5): 741-748.

Holten, D. and J.J. Van Wijk, 2009. Force-directed edge bundling for graph visualization. Comput. Graph. Forum, 28(3): 983-990.

Holy, L., K. Jezek, J. Snajberk and P. Brada, 2012a. Lowering visual clutter in large component diagrams. Proceeding of International Conference on Information Visualization. IEEE Computer Society, Washington, DC, USA, pp: 36-41.

Holy, L., J. Snajberk and P. Brada, 2012b. Lowering visual clutter of clusters in component diagrams. Proceeding of International Conference on Software Engineering Advances, IARIA. Red Hook, NY, USA, pp: 304-307.

Holy, L., J. Snajberk and P. Brada, 2012c. Evaluating component architecture visualization tools-criteria and case study. Proceeding of GRAPP/IVAPP, SciTePress, pp: 737-742.

IBM Rational Software Architect, 2014. Retrieved form: http://www.ibm.com/developerworks/rational/products/rsa/. (Accessed on: Sep. 29, 2014)

Johnson-Laird, P.N., 1983. Mental models: Towards a Cognitive Science of Language, Inference, and Consciousness. Harvard University Press, Cambridge, MA.

Laidlaw, D.H., J.S. Davidson, T.S. Miller, M. da Silva, R.M. Kirby, W.H. Warren and M. Tarr, 2001. Quantitative comparative evaluation of 2d vector field visualization methods. Proceeding of the Conference on Visualization (VIS '01). Washington, DC, USA, pp: 143-150.

MacKenzie, I.S., 2012. Human-computer Interaction: An Empirical Research Perspective. Elsevier Science, Burlington.

Magicdraw, 2014. Retrieved form: http://www.nomagic.com/ (Accessed on: Sep. 29, 2014)

Mancoridis, S., B.S. Mitchell, C. Rorres, Y. Chen and E.R. Gansner, 1998. Using automatic clustering to produce high-level system organizations of source code. Proceeding of the 6th International Workshop on Program Comprehension (IWPC '98). Washington, DC, USA, pp: 45.

McGee, F. and J. Dingliana, 2012. Visualising small world graphs-agglomerative clustering of small world graphs around nodes of interest. In: Richard, P., M. Kraus, R.S. Laramee and J. Braz (Eds.), GRAPP & IVAPP, 2012: Proceeding of International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications. SciTePress, Rome, Italy, pp: 678-689.

Meyer, J., J. Thomas, S. Diehl, D.A. Keim and B. Fisher, 2010. From visualization to visually enabled reasoning. In: Hagen, H. (Ed.), Scientific Visualization: Advanced Concepts. Vol. 1 of Dagstuhl Follow-Ups, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp: 227-245.

Moody, D. and J. van Hillegersberg, 2009. Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams. In: Gasevic, D., R. Lammel, E. Van Wyk (Eds.), SLE, 2008. LNCS 5452, Springer-Verlag, Berlin, Heidelberg, pp: 16-34.

Morris, S. and G. Spanoudakis, 2001. UML: An evaluation of the visual syntax of the language. Proceeding of the 34th Annual Hawaii International Conference on System Sciences. Washington, DC, USA, pp: 10.

Nuxeo, 2014. Retrieved form: http://www.nuxeo.com/ (Accessed on: Sep. 29, 2014)

Object Management Group, 2009. UML Superstructure Specification (2009).

OpenWMS, 2014. Retrieved form: http://www.openwms.org/ (Accessed on: Sep. 29, 2014)

OSGi Alliance, 2009. OSGi Service Platform v4.2. Core Specification, OSGi Alliance (2009).

Plugin, 2014. Dependency Visualization. Retrieved form: http://www.eclipse.org/pde/incubator/dependency-visualization/. (Accessed on: Sep. 29, 2014)

Rafiei, D., 2005. Effectively visualizing large networks through sampling. Proceeding of IEEE Visualization (VIS, 2005), pp: 375-382.

Rausch, A., R. Reussner, R. Mirandola and F. Plasil, 2008. The Common Component Modeling Example: Comparing Software Component Models.1st Edn., LNCS 5153, Springer-Verlag, Berlin, Heidelberg, pp: 1-3.

Sensalire, M., P. Ogao and A. Telea, 2009. Evaluation of software visualization tools: Lessons learned. Proceeding of 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT, 2009), pp: 19-26.

Snajberk, J., L. Holy and P. Brada, 2012. Comav-a component application visualisation tool. Proceeding of International Conference on Information Visualization Theory and Applications, SciTePress.

Softvision, 2014. Retrieved form: http://www.win.tue.nl/vis1/home/lvoinea/soft/SoftVisionManual.pdf. (Accessed on: Sep. 29, 2014)

Sun Microsystems, 2006. Enterprise Java Beans (TM). Version 3.0, EJB Core, Sun Microsystems (2006).

Szyperski, C., 2002. Component Software: Beyond Object-oriented Programming. 3rd Edn., Addison-Wesley, ACM Press.

Telea, A. and L. Voinea, 2004. A framework for interactive visualization of component-based software. Proceeding of the 30th EUROMICRO Conference. IEEE Computer Society, Washington, DC, USA, pp: 567-574.

Visual Paragigm, 2014. Retrieved form: http://www.visual-paradigm.com. (Accessed on: Sep. 29, 2014)

Wettel, R. and M. Lanza, 2007. Visualizing software systems as cities. Proceeding of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis. Society Press, pp: 92-99.