An Advanced Interactive Visualization Approach for Component-Based Software: A User Study

Jaroslav Šnajberk, Lukas Holy, Kamil Jezek and Premek Brada
Department of Computer Science and Engineering, Faculty of Applied Sciences,
University of West Bohemia, Pilsen, Czech Republic
{snajberk, lholy, kjezek, brada}@kiv.zcu.cz

***Abstract*—We present a user study that compares user performance during architectural analysis using two different approaches to visualizing component-based applications structure: AIVA (Advanced Interactive Visualization Approach) and UML (Unified Modeling Language). AIVA is a research proof of concept focused on extensive use of interactivity in visualization of structure. UML is an industrial standard in the field of software visual modeling. Participants of this user study tested how fast they could perform six basic tasks which were selected so as to gain understanding of component dependencies in a medium-sized OSGi application. The results show that AIVA helps to find answers on average three times faster than UML. The study and its results provide a quantitative support for our hypothesis that the structure of component-based applications should be visualized interactively using dedicated notation rather than in static UML diagrams to improve understanding of the whole application architecture.**

Keywords-software visualization; component; UML; user study; interactivity.

I. INTRODUCTION

Component-based software engineering is a modern software discipline that encapsulates objects into black box structures called components to maximize re-usability and to improve logical composition of the application. Although the concept of a component is not new, approaches able to fully visualize the structure of these applications are lacking. However, such visualization is important for engineers to thoroughly understand the applications.

AIVA (Advanced Interactive Visualization Approach) [1] is our research visualization approach that is built on the idea that interactivity is beneficial for the study of structure and that different interactive techniques should be adopted to maximize the impact of interactivity. This idea is in contrast with the commonly used static approach of standardized UML [2] and its component diagrams. Such an interactive approach should be able to describe any component in at least the same level of detail as UML can, without introducing more occlusion in the diagrams. Interactivity should lead to a simplification of structure visual representation, especially when combined with techniques that are able to provide all details to the user just when he needs them.

To validate the approach and assess its practical implementation, we have performed a controlled user study focused on performance (time required to provide a correct answer) during architectural analysis tasks. This paper describes the whole study in terms of its design and results, together with necessary background information on the AIVA approach and further discussion of the results.

A. Current State of the Art

The Unified Modeling Language (UML) provides three groups of diagrams to model both static and dynamic features of software [2], including the component diagram. The notation used in this diagram, however, captures components only on a general level, while their details differ greatly between component models, both commercial and research ones – OSGi [3], EJB [4], SOFA 2 [5] and others.

A component model can further introduce its own unique features like behavior protocols [6] or hierarchical decomposition. To this end, UML supports user-defined profiles that are able to model most – but not all – of the features satisfactorily; although, their visual representation tends to be difficult to read. An opposite approach is sometimes used, namely to augment a given component model with its own visual notation as, for example, in the case of SaveCCM [7]. A brief study of the currently used approaches and tools is provided by Holy [8].

Even when a UML profile is complemented with a tool providing good visualization of the profile, an essential problem still remains: that the structure of component applications tends to be more complex than most class structures. Contracts between components often involve several features like event queues, provided interfaces, imported packages, etc., leading to many diagram lines per component pair. The resulting structure is therefore more complex and harder to read.

B. Related Work

Research efforts related to our visualization of software architectures fall in two broad categories: displaying the structure and dealing with interactivity. The efforts to display these structures are most commonly oriented on extensions of the UML itself, without taking interactivity under consideration; while Dumoulin [9] introduces layers to support multiple views in one diagram, Byelas [10] suggested the use of colored areas of interest to improve orientation in classical UML component diagrams. Other works are even less relevant to the work presented in this paper.

Telea's [11] work on (interactive) visualization of component-based software is generic and mostly similar to

 ISBN: 978-1-61208-230-1

the work presented in this paper, but it does not provide many details about components themselves and can hardly be compared with UML. Wettel and Lanza [12] visualize software as cities – they define three dimensions: two are used as the base of a building and the third is used as the height of a building. This approach could be easily used on component software, but again it does not provide details needed to get full comprehension of the 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 lighten the cognitive load, namely hide unnecessary details, as Ric Holt highlighted with several examples in [13]. The importance of interactivity for the ability to make decisions about a mental model is mentioned in several studies, one of which is Meyer et al. [14]. He goes even further and defines a new science of visually enabled reasoning, implying that interactivity is its key enabler.

Finally, works concerned with the evaluation of new information visualization approaches are also related, as we studied them prior to designing our own user study. Therefore, the work of Camilla Forsell [15] should be highlighted, as it provides a clear guide for similar studies. Laidlaw [16] uses a similar comparative study of performance on 2D vector field visualization methods. Evaluation of software visualization was also described by Sensalire et. al. in [17], cooperating with Telea, mentioned earlier.

C. Structure of the Text

This paper first describes AIVA and UML in Section II in order to provide sufficient understanding of them. Section III describes the design of the presented user study in detail. Technical information related to the hardware and specific software technologies used are provided in Section IV.

Section V presents the results of the performed user study, whose conclusions are then discussed in Section VI. Finally this paper is concluded by Section VII with a summary of findings.

II. Overview of Compared Approaches

This section discusses both AIVA and UML, where the former is described in greater detail, as it is an experimental approach which is not commonly known. UML is touched only briefly, being a common standard. Special care is given to the techniques and features that were used by participants in the study. Both approaches were already compared in a case study [18] which focused solely on the readability of the diagrams in these two approaches – compared to their performance testing, which is the subject of this paper.

A. AIVA

The Advanced Interactive Visualization Approach (Project page: `http://www.assembla.com/spaces/comav`) uses an oriented graph to visualize components and their relations. Its visual notation is unified for all component models (including the hierarchical ones) and is partially similar to that of a UML class diagram. However, it differs in the representation of the list of elements inside the component "box" – AIVA prefers hierarchical ordering based on the characteristics of elements. The resulting groups of elements stand each for itself to provide better orientation. A thorough description of AIVA was published in [1], so below we discuss mainly its interactive features that can help increase user performance, in line with the goal of this paper.

The navigation and explore features that are most frequently used when studying the diagram accommodate these interactive techniques: scrolling, zooming, panning, outline view and quick search (move the view on the diagram to show the component selected in the project overview). AIVA improves the zooming technique: when zoomed out, standard zooming simply changes the scale of the standard output, which makes the details of a component unreadable and thus useless. AIVA in this case hides the details and enlarges only the information that is always important – the name of the component. This change should improve orientation in the diagram.

Highlighting helps to further improve orientation in the diagram. Almost any interaction will highlight the subject of interaction with bright, easily distinguishable color in both the diagram and the overview. One click on a component highlights the component; one click on a connection line highlights the line together with both components connected by the line and the elements involved in this relation. Double click on an element highlights all connection lines related to this element as well as all connected components.

Decreased complexity of the diagram is yet another way to improve diagram readability; AIVA uses information-hiding techniques that help to achieve this goal, together with details on demand (discussed next). A very effective complexity reduction method is to collapse the connection lines, since the reduction of the number of lines in the diagram makes it less complex and easier to read; in AIVA, therefore, there is only one connection line between each pair of connected components. In addition, the information labels identifying the connection type and connected elements are by default hidden. Finally, interfaces and other component interface features are not diagram nodes and the structure is therefore less complex.

Details on demand are used to access the hidden information. When one clicks on the connection line, an information box appears showing a detailed list of all connected elements; therefore, no information value is lost. Additional details are also provided when the user clicks on the component, revealing information about, e.g., its version, license, symbolic name, etc. Similar information about every component element is accessible by hovering over its name.

Other features that are supported by AIVA are used mostly to offer a different view on the same thing. However, these features are not in the scope of this paper, so we mention them only to present AIVA completely:

1) Grouping and filtering, based on the characteristics of the elements. Support for user defined sets of groups.
2) Conditional formatting able to work with component-

 ISBN: 978-1-61208-230-1

model-specific information and various secondary data that are normally hidden.
3) Reconfiguration features, namely change of representation of components or diagram layout.

B. UML

Unified Modeling Language on its own is a static approach in which the diagrams look the same on a computer screen and on paper. For full component modeling support, user profiles are necessary, which considerably shortens the list of available tools. The standard interactive techniques used across all these UML tools are navigation ones – scrolling, panning, zooming, and overview. The overall usability of UML component diagrams is related more to the specific features of concrete tools than to the UML notation itself. Therefore, tools with some "added value" should be selected to objectively investigate UML usability.

IBM Rational Software Architect (RSA) can be considered as such an advanced UML tool. Besides all standard navigation features, it 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. Added value is in its "*properties view*", displayed at the bottom of the screen. This view shows all the details about components and relations and, most importantly, it 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 the component. This list clearly specifies which kind of relation is used and which component is related. The name of the related component has the form of a link, so the user can easily find more information about it.

III. Design of the Study

This section provides the details about the goal and mechanics of the performed user study.

A. Goal of the Study

The main goal of this study is to evaluate the performance of users during architecture analysis in two different approaches – UML and AIVA. The hypothesis tested was: "It is faster for engineers to study the structure of component-based applications interactively rather than using static diagrams." The null hypothesis was that studying component-based applications' structure is comparably fast when using interactive and static visualization methods.

The results of this user study can therefore help in finding out to what degree interactivity is useful. These questions are important because the level of interactivity used in AIVA is high and could negatively affect the user's performance while he collects some more detailed information, specifically because a lot of this information is hidden and revealing it requires user interaction.

The set of tasks used in the study simulates the activities performed during one step of architecture analysis. These tasks are focused on collecting knowledge about one component – its features, dependencies and overall context consisting of related components. When analyzing the whole architecture, one needs to repeat this step for most of its components. The concrete set of tasks is discussed thoroughly further below.

B. Profile of Participants

The structure of component-based applications is studied by software engineers who work on these applications. They have a deep knowledge of components and UML to be able to understand the diagram presented. Such people are hard to get to participate in a user study that takes at least one hour; thus we decided to ask our colleagues to participate. Use of academics and Ph.D. candidates was encouraged by Sensalire et. al. in [17], based on their lessons learned: "They are willing to take part in studies for the sake of gaining knowledge and may require less or no additional motivation", while still clearly being professionals in the field of software engineering.

The participants were young software engineers selected from different groups at our department. They were confident in most UML diagrams; their knowledge of component diagrams was tested specifically before participation. Most of the participants use components on a daily basis and the rest were briefly trained before the study. All of the participants were confident in the required basics of component-based development before undertaking the questions.

All participants were also trained in both tools that were used to test the two approaches: UML/RSA and the AIVA research prototype. First, the tool was presented to them. We shared our working experience on how to get various types of information effectively. Then every participant had unlimited time to test all types of tasks that he would encounter. Participants were handled individually and guidance was given when asked. The training ended when the participant felt confident and able to perform all types of tasks used in this user study.

C. How the Study Was Performed

The process repeated with every participant was as follows:

1) Verification of knowledge
2) Training in Tool 1
3) Performing all tasks in Tool 1
4) Training in Tool 2
5) Performing all tasks in Tool 2

Verification of UML and component knowledge took about 20 minutes to ensure the participant's expertise. Training in both tools took about 40 minutes, until the participant felt confident. All tasks were performed in under 10 minutes, because the tasks were quite short.

All participants were observed for the whole time of the study and there were no interruptions nor any advice while they searched for the answers to a given task. The time was measured from the moment the question was read and understanding was confirmed by the participant, to the point when a correct and full answer was given. We required users to visually verify the information as part of the answer, i.e. to pinpoint the found information in the diagram.

 ISBN: 978-1-61208-230-1

All participants were divided into two groups of the same size. Group #1 started with AIVA and group #2 with UML. The set of tasks was identical for both approaches; however, the tested approaches provide such different diagrams that the participants could not gain an advantage by performing the same tasks again with the other tool. Moreover, training in the second tool was performed after testing first tool to distract participants from the tasks. During the training phase, similar tasks were practiced by the participants on different components in a completely different application to ensure that they could not gain any knowledge related to the test tasks.

The study was intentionally designed this way to prevent bias of participants which could erode the validity of results.

IV. Technical Background

This section provides technical details required to recreate the user study described. An overview of the OSGi component model and CoCoME application used is provided, after which the set of tasks (bound to the application) is described in its final form. Finally, details about hardware are presented to provide the whole picture.

A. OSGi Component Model

OSGi [3] is a multi-platform Java component solution focused on dynamic deployment and assembly of components. OSGi components are black-boxes; their nature and features are described in a manifest file. The communication between components is realized by *services* which are implementations of interfaces, thus keeping their black-box nature.

Apart from services, both provided and required, these components can depend on *Imported packages* and other components – *Required bundles*. Thus there are three completely different types of relations in the OSGi environment. Complex analysis of OSGi and its key features is described in [19], where the author suggests an OSGi profile for the ENT meta-model which is used by AIVA.

After a thorough study of this profile, we developed a similar one for UML, so it can model the same information as the ENT meta-model. We already described this profile in our previous study [18], which was more concerned with the question of how this information is visualized.

B. CoCoME

CoCoME stands for Common Component Modeling Example [20]. It models an information system for supermarket chains and is used for the purpose of comparing different approaches to component-based software development. It has been officially implemented in 13 component models; we created an OSGi implementation (`http://www.assembla.com/spaces/comav/documents/tag/CoCoME`).

The CoCoME application consists of three main parts. First is a Cashdesk part, which contains the cashdesk, including barcode scanners, credit card readers, etc. The second part is a store backoffice server and a store client. Finally, the chain part consists of an enterprise server and client applications. CoCoME is assembled from 37 components using 12 interfaces, thus representing a medium-size application. The complete diagram of the whole application is accessible in both AIVA and UML forms on our project page (`http://www.assembla.com/spaces/comav/wiki/Comparison_of_AIVA`).

C. Tasks

The tasks described below were tested on the CoCoME application. Because of this, they are formulated directly for its components; however, they can be easily generalized. The tasks are basic and contribute to answering one complex question: how is a particular component (*cocome-osgiDS-store.impl*) integrated in the CoCoME application. One has to find out what this component offers and requires and uncover its ties to other components, simulating the activities performed during one step of the architecture analysis. The most complex component of the application was chosen for these tasks.

The tasks were identified based on our experience with the structure of component-based applications and hints obtained during interviews with several software engineers from local software companies. The exact wording was then designed to cover all aspects of one concrete component.

- Q1. Which packages are imported by component *cocome-osgiDS-store.impl*?
- Q2. Which elements of component *cocome-osgiDS-store.impl* are unused (i.e. have no relationship)?
- Q3. Which components use the service *StoreIf* provided by *cocome-osgiDS-store.impl*?
- Q4. Which components depend on *cocome-osgiDS-store.impl*?
- Q5. Which components are required by *cocome-osgiDS-store.impl*?
- Q6. Which elements does *cocome-osgiDS-store.impl* need from *cocome-osgiDS-data*?

D. Hardware

Computer hardware did not influence the results of the study since the bottleneck for performance was the user's ability to interact and read the information from the diagram. However, to provide complete technical background, here are the specifications of the testing computer: Intel Core i5 3Ghz CPU, 4GB DDR3 1066Mhz RAM, 7200RPM HDD and, most importantly, 24" LCD with 1920x1080 resolution. This computer proved to be fast enough to ensure a comfortable working experience and the screen resolution was sufficient for visualization purposes.

V. Results

This section provides detailed results of this study for each approach and their comparison. As the reader may note, the results differ greatly depending on the participant. This was caused by individual perception, orientation abilities and how quickly they were able to click the mouse. (A lot of attention was paid to preparing all of them thoroughly, see Section III.)

 ISBN: 978-1-61208-230-1

Twelve users participated in the study, identified as A-L in the tables below. The last two participants (K and L) are co-authors of this paper and mentored the rest of the participants. Our performance is listed in the results to show the peak performance of the tasks, as we knew exactly what we were looking for and how to retrieve this information. We followed the same rules as any other participants and accepted the answer only after visual confirmation. Our results are not used in later statistics. The Q1-Q6 identifiers in the result tables refer to the tasks from Section IV-C.

Statistical tables present important statistical values calculated per question. The "Total" column provides the sum of values per row, calculated from results data with millisecond precision (thus, the simple sum of provided values may differ).

A. Performance in AIVA

Results of all participants are presented in Table I, while statistical values are in Table II. The biggest strength of AIVA was the search for unused elements (Q2), as it provides the answer immediately and most participants were able to read it right away; however, a few of them did not at first understand this information. The biggest weakness was finding the dependent components (Q4), because most of the participants forgot to read the type of arrow indicating the type of the connection and got a little confused – searched for the answer elsewhere before they found it. As a result, the time needed for a correct answer was longer, as we waited for them to solve the problem themselves.

TABLE I
RESULTS OF USERS IN AIVA [MIN:SEC].

ID	Q1	Q2	Q3	Q4	Q5	Q6	SUM
A	0:46	0:11	0:20	0:32	0:33	0:28	2:50
B	0:16	0:09	0:42	0:25	0:35	0:25	2:32
C	0:22	0:08	0:18	1:02	0:23	0:27	2:40
D	0:51	0:33	0:20	0:41	0:44	0:50	3:59
E	0:12	0:10	0:11	1:20	0:22	0:10	2:25
F	0:25	0:09	0:23	0:27	0:31	0:38	2:33
G	0:23	0:22	0:19	0:40	0:23	0:29	2:36
H	0:29	0:06	0:16	0:37	0:29	0:16	2:13
I	0:07	0:04	0:08	0:24	0:16	0:24	1:23
J	0:15	0:24	0:17	0:31	0:25	0:17	2:09
K	0:08	0:03	0:08	0:13	0:19	0:10	1:01
L	0:12	0:04	0:08	0:15	0:17	0:11	1:07

TABLE II
STATISTICS OF USERS IN AIVA.

Measure	Q1	Q2	Q3	Q4	Q5	Q6	Total
Avg	0:24	0:13	0:19	0:39	0:28	0:26	2:32
Median	0:22	0:09	0:18	0:34	0:27	0:26	2:18
Min	0:07	0:04	0:08	0:24	0:16	0:10	1:09
Max	0:51	0:33	0:42	1:20	0:44	0:50	5:00
Std dev.	0:13	0:08	0:08	0:17	0:07	0:10	N/A

B. Performance in RSA

Results of all participants are presented in Table III, while statistical values are in Table IV. The biggest strength of RSA was looking up service clients (Q3), as it provided the answer almost immediately, while the biggest weakness was finding the dependent components (Q4) due to worse RSA support in connecting components through ports. Participants gave a stable performance as they are familiar with UML notation. The graphical user interface of RSA is more user friendly, which also helped users in orientation. Often, Participants were delayed by accidental clicking on the connection line – RSA had centered the screen on it and they lost the context of the studied component.

TABLE III
RESULTS OF USERS IN RSA [MIN:SEC].

ID	Q1	Q2	Q3	Q4	Q5	Q6	SUM
A	1:04	2:40	0:12	2:56	2:43	2:10	11:45
B	1:22	2:06	0:11	1:40	2:39	2:08	10:06
C	1:13	2:30	0:25	1:58	2:49	2:14	11:09
D	1:19	1:30	0:27	1:23	2:25	2:10	9:14
E	0:36	0:59	0:17	0:43	1:41	1:00	5:16
F	1:24	1:05	0:21	1:01	1:40	2:49	6:12
G	0:43	0:30	0:07	0:39	1:46	1:20	5:05
H	0:46	1:14	0:09	0:54	2:17	0:52	6:12
I	0:52	0:34	0:08	0:28	1:00	0:36	3:38
J	0:59	1:06	0:21	0:40	1:48	1:28	6:22
K	0:32	0:42	0:10	0:34	1:07	0:46	3:51
L	0:39	0:52	0:11	0:25	0:54	0:39	3:40

TABLE IV
STATISTICS OF USERS IN RSA.

Measure	Q1	Q2	Q3	Q4	Q5	Q6	Total
Avg	1:01	1:25	0:15	1:14	2:04	1:40	7:42
Median	1:01	1:10	0:14	0:57	2:02	1:48	7:14
Min	0:36	0:30	0:07	0:28	1:00	0:36	3:17
Max	1:24	2:40	0:27	2:56	2:49	2:49	13:05
Std dev.	0:16	0:43	0:06	0:43	0:33	0:41	N/A

C. Comparing the Results

Apart from the measured values, a useful piece of information resulting from this study is the performance ratio of AIVA to RSA. Comparing this ratio for every participant can bring more insight than comparing the global numbers. The highest ratio was for participant A, who was 4.15 times faster in AIVA than in RSA. The lowest ratio was for participant G, only 1.96 times faster in AIVA than in RSA. The rest of the participants were within these extremes; however, they were on average 3 times faster in AIVA – the average test time in RSA was 462 seconds, compared to 152 seconds in AIVA.

The average results are compared with the standard deviation in Figure 1. Normal distribution says that 70% of users would fall within these limits. This figure clearly shows that AIVA was faster in tasks Q1, Q2, Q4, Q5 and Q6, that is in 83% of cases, while it was slower in task Q3, which was the strongest task in RSA.

Figure 2 comprehensibly presents minimum, maximum and median values in a comparable way, so that these values can be conveniently studied in one place.

Lastly, Figure 3 contrasts the longest times measured in AIVA with the shortest times measured in RSA. This figure

 ISBN: 978-1-61208-230-1

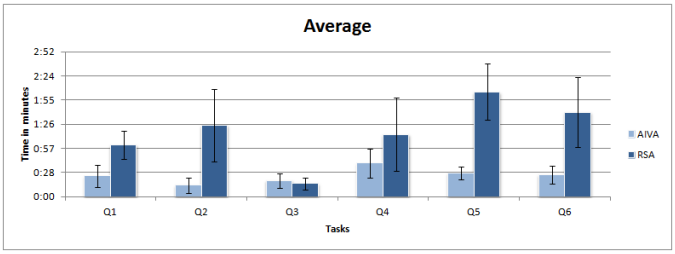

Fig. 1. Comparison of average results.

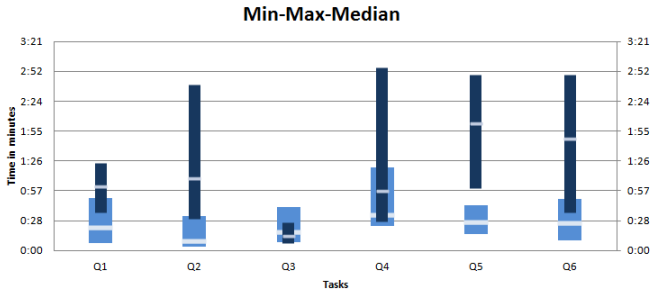

Fig. 2. Minimum and maximum values with median marked.

presents a different look at these extreme values, showing how a poor use of AIVA compares with best-performing RSA users. The numbers show that even in this case, AIVA is comparable in two thirds of tasks; RSA has its best results significantly faster in tasks related to service dependencies.

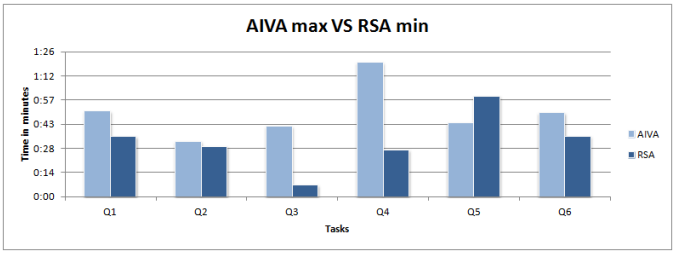

Fig. 3. Comparison of maximal AIVA results with minimal RSA results.

Two scenarios were tested to compare results in more depth. The first scenario tested if users who perform best in UML are also very fast in AIVA. All participants were ordered from fastest to slowest in both AIVA and UML and their order was compared. Four participants from the UML top 5 were also in the AIVA top 5. The notes on the remaining participant who did worse in AIVA showed that he was overconfident because of his expertise in UML. He started the test in AIVA and had to think longer about how to finish the given tasks. As a result of this scenario, it is possible to conclude that good analysts will benefit from using AIVA.

The second scenario tested where in the distribution are users who were really slow in UML. All participants were ordered from fastest to slowest in UML and, also, their performance ratio between AIVA and UML was ordered from highest to lowest. Four participants from the bottom 5 in UML were among the top 5 users overall in the performance ratio. (The one who was not in the top 5 also had significantly worse results in AIVA – he also felt confident in AIVA although it turned out he should have kept training for some more time.) The five slowest participants in UML were on average 3.5 times faster in AIVA, while the top 5 UML participants were on average only 2.5 times faster. It is possible to conclude that casual users of UML will benefit the most from using AIVA.

VI. Discussion

The previous section provided results of this user study and compared them. By studying these results it is possible to conclude several things on which this section comments. It is important to realize that AIVA did trade-off some characteristic features of UML to get these better results. The most important trade-offs are that AIVA can not be used on paper; it is usable only for component-based applications and, moreover, only for the structure of these applications.

A. Measures of AIVA Performance

First of all, the time required to finish a task in AIVA is more consistent – the standard deviation in RSA (48 seconds) is almost four times higher that that of AIVA (14 seconds). The reason is that UML itself and thus also RSA has different levels of recognition and therefore handling for different elements – work is really fast with some (tasks that depend mostly on interfaces) but slow with others (tasks that depend mostly on ports). On the other hand, AIVA provides the same level of support for all types of elements on both visual and interaction levels.

The previous conclusion leads to a more important one – the choice of tasks is not so important for AIVA as it is for UML. In other words, AIVA should be able to provide stable user performance for any task set, in any component model. In contrast, a user's performance in UML depends on the selected tasks, the selected UML tool and the component model.

Task Q4 (searching for clients of the studied component) was the slowest one in both approaches. The reason is that the component was widely used by many other components in the CoCoME application. Therefore, it took time to find them all.

One should also look at the fastest task for UML – Q3, which worked with dependencies of interfaces in a tool which is able to list all these dependencies at once. This can be recognized as a best case UML scenario, with the fastest time of 7 seconds, while AIVA required 8 seconds. Median values are 14 seconds for UML and 18 seconds for AIVA – that is, AIVA is 28% slower. The worst case scenario for UML would be Q5, which worked a lot with dependencies of packages (ports). The fastest user finished this task in 16 seconds in AIVA but took 60 seconds in UML. Median values are 27 seconds in AIVA and 122 seconds in UML – UML is 450% slower. These numbers again indicate that AIVA would be faster in any mixed task set.

From the results provided, it is thus possible to conclude that the level of interactivity used in AIVA is useful and that

 ISBN: 978-1-61208-230-1

the null hypothesis is not valid. Interactivity helped AIVA to provide a simpler diagram, so users could orientate themselves easier, and the overall user performance was better even when the interaction was required to gather the necessary information.

B. Participant Opinions

This user study confirmed that AIVA is faster than UML, but we also asked the participants a few subjective questions after they finished:

1) Do you consider the AIVA or UML diagram clearer? Why?
2) Which was more comfortable to work in, AIVA or RSA? Why?
3) Do you have other suggestions?

All participants answered that AIVA provides a clearer diagram that is more readable and understandable. They mentioned these reasons: fewer lines, hidden details on zoom, all information in one place and very readable structure of elements.

One participant felt more comfortable in RSA because labels were always visible and a click on lines centered the screen. The rest of the participants felt more comfortable in AIVA, giving these reasons: clearer GUI, packages shown inside components, much faster operation, information easier to reach, better interactive overview. These participants also did not like the RSA feature that centered the screen after they clicked on a line because it happened often by accident and they lost thecontext.

VII. Conclusion and Future Work

This paper described a complete user study that compared performance in component architecture analysis tasks using two different component visualization approaches – AIVA and UML. AIVA is implemented as a research proof of concept, while UML is supported by a lot of commercial tools. Rational Software Architect was chosen to represent these tools because of its ability to easily study relations, which was most needed in this experiment.

The data obtained show that users working interactively (i.e. in AIVA) are approximately three times faster than those using UML. In only one of six tasks was UML faster, while AIVA performed better in the remaining 5/6 of tasks. The discussion section above provides insight into the reasons and on how different tasks could affect the overall performance.

Results of this user study therefore confirm that advanced visualization of component-based application architecture using a high level of interactivity is beneficial for users. Even the increased interaction required to uncover hidden information does not introduce significant problems.

Our future work will evaluate if AIVA can be used by software engineers in real-life scenarios. In particular, we want to test if it helps users to understand the application structure starting from the beginning of the learning process to the point where they have sufficient insight to make decisions and answer complex questions.

Acknowledgment

The work was supported by the University of West Bohemia grant SGS-2010-028 Advanced Computer and Information Systems.

References

[1] J. Snajberk and P. Brada, “Interactive Component Visualization,” in *Proceedings of International Conference on Evaluation of Novel Approaches to Software Engineering*. SciTePress, 2011, pp. 218–225.

[2] Object Management Group, “UML Superstructure Specification,” Object Management Group, OMG Specification formal/2009-02-02, 2009.

[3] OSGi Alliance, “OSGi Servise Platform Core Specification,” OSGi Alliance, OSGi Specification, 2009.

[4] Sun Microsystems, Inc., “Enterprise JavaBeans(TM) Specification,” Sun Microsystems, Inc., SUN Specification, 2001.

[5] T. Bures, P. Hnetynka, and F. Plasil, “SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model,” in *SERA*. IEEE Computer Society, 2006, pp. 40–48.

[6] F. Plasil and S. Visnovsky, “Behavior Protocols for Software Components,” *IEEE Trans. Software Eng*, vol. 28, no. 11, pp. 1056–1076, 2002.

[7] H. Hansson, M. Akerholm, I. Crnkovic, and M. Tarngren, “SaveCCM - A Component Model for Safety-Critical Real-Time Systems,” in *EUROMICRO*. IEEE Computer Society, 2004, pp. 627–635.

[8] L. Holy, J. Snajberk, and P. Brada, “Evaluation Component Architecture Visualization Tools,” in *Proceedings of International Conference on Information Visualization Theory and Applications*. SciTePress, 2012.

[9] C. Dumoulin and S. Gerard, “Have Multiple Views with one Single Diagram! A Layer Based Approach of UML Diagrams,” Institut National de Recherche en Informatique et en Automatique, Universite des Sciences et Technologies de Lille, Research report INRIA-00527850, October 2010.

[10] H. Byelas, E. Bondarev, and A. Telea, “Visualization of areas of interest in component-based system architectures,” in *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 160–169.

[11] A. Telea and L. Voinea, “A Framework for Interactive Visualization of Component-Based Software,” in *Proceedings of the 30th EUROMICRO Conference*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 567–574.

[12] R. Wettel and M. Lanza, “Visualizing software systems as cities,” in *In Proc. of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. Society Press, 2007, pp. 92–99.

[13] R. Holt, “Software Architecture as a Shared Mental Model,” in *Proceedings of International Workshop on Program Comprehension*, 2002.

[14] J. Meyer, J. Thomas, S. Diehl, B. Fisher, and D. A. Keim, “From Visualization to Visually Enabled Reasoning,” in *Scientific Visualization: Advanced Concepts*, ser. Dagstuhl Follow-Ups, H. Hagen, Ed. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010, vol. 1, pp. 227–245.

[15] C. Forsell, “A guide to scientific evaluation in information visualization,” in *Information Visualisation (IV), 2010 14th International Conference*, july 2010, pp. 162 –169.

[16] D. H. Laidlaw, J. S. Davidson, T. S. Miller, M. da Silva, R. M. Kirby, W. H. Warren, and M. Tarr, “Quantitative comparative evaluation of 2d vector field visualization methods,” in *Proceedings of the conference on Visualization ’01*, ser. VIS ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 143–150.

[17] M. Sensalire, P. Ogao, and A. Telea, “Evaluation of software visualization tools: Lessons learned,” in *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*, 2009, pp. 19 –26.

[18] J. Snajberk, L. Holy, and P. Brada, “AIVA vs UML: Comparison of Component Application Visualizations in a Case-Study,” in *Proceedings of 16th International Conference on Information Visualization*, 2012.

[19] L. Valenta and P. Brada, “OSGi Component Substitutability Using Enhanced ENT Metamodel Implementation,” Department of Computer Science and Engineering, University of West Bohemia, Tech. Rep. DCSE/TR-2006-05, 2006.

[20] A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, *The Common Component Modeling Example: Comparing Software Component Models*, 1st ed. Springer Publishing Company, Incorporated, 2008.

 ISBN: 978-1-61208-230-1