VALCRI
VISUAL ANALYTICS FOR SENSE-MAKING IN CRIMINAL INTELLIGENCE ANALYSIS

***VALCRI* WHITE PAPER SERIES**
VALCRI-WP-2017-001
1 January 2017
Edited by B.L. William Wong

Architecture, Development and Testing Environment for a Visual Analytics-based Criminal Intelligence Analysis System

Rani Pinchuk[1], Nick Evers[1], Christophe Vandenberghe[1].
Patrick Aichroth[2], Rudolf Schreiner[3], and B.L. William Wong[4]

[1]Space Applications Services NV/SA
Leuvensesteenweg, 325,
1932 Zaventem,
BELGIUM

[2]Fraunhofer Institute for Digital Media Technology
Ehrenbergstraße 31
98693 Ilmenau
GERMANY

[3]Object Security, Ltd
St John's Innovation Centre
Cowley Road
Cambridge CB4 0WS
UNITED KINGDOM

[4]Middlesex University London
The Burroughs, Hendon
London NW4 4BT
UNITED KINGDOM

Project Coordinator
Middlesex University London
The Burroughs, Hendon
London NW4 4BT
United Kingdom.

Professor B.L. William Wong
Head, Interaction Design Centre
Faculty of Science and Technology
Email: w.wong@mdx.ac.uk

EUROPEAN COMMISSION

ABSTRACT

The VALCRI architecture is built from different Docker containers that speak with each other using mostly REST interfaces. The architecture is designed to incorporating Security, Ethics, Privacy and Legal (SEPL) solutions. The data stores – the Unstructured Database (UDB) and the Structured database (SDB) – used are controlled by SEPL Enforcement components and a Template Engine manages the previously checked and accepted query templates that can be sent to the data stores. The Advanced User Interface (AUI) server is also designed with SEPL in mind: a Jetty (Java HTTP server and Java Servlet container) instance is created per user by a Jetty Lifecycle Management component. Each such instance lives inside a Docker container to promote isolation. The Jetty instance hosts the mid-tier services, serves the front-end JavaScript code and manages the communication between the mid-tier services and the front-end. The midtier services are written in java and front-end components are implemented using GWT – Google Web Toolkit. The Model View Presenter (MVP) design pattern is implemented. The SEPL Enforcement components, as well as the analysis components, and the Jetty Containers communicate to the CAS (Central Authentication Service) component to ensure that only authorized users can perform certain actions and allows data to be properly restricted. The user management of the CAS component is linked with the LDAP (Lightweight Directory Access Protocol) component to manage and authenticate user credentials. All the components may use the interface exposed by the GrayLog component, ensuring that any action done by any component can be logged in the logging storage.

Systematic system integration is a key principle for the success of VALCRI and for that a development pipeline was designed. This pipeline aims to provide continuous system integration while promoting collaboration, contributions, quick feedback to contributions, changing and evolving interfaces, and above all respecting the principle of “keep it working” – allowing to introduce many contributions in small steps while the system continues to compile and work. The GIT Source Control Management (SCM) platform is used for keeping track of changes in the source code. It is accompanied by a GitLab installation that provides a user interface that allows managing the user accounts and the individual code repositories. In order to compile and build the code available from the SCM, a custom build system was developed using Gradle. The Gradle setup is also accompanied with a Nexus component – an artefact repository – which hosts all the compiled, binary VALCRI system components. For each change in the SCM, the source code is automatically rebuilt and all tests are run. In order to do this, a Jenkins installation was put into place. Because the VALCRI architecture includes many services working on different environments, Docker was selected to allow building, shipping and running the complete environments.

Keywords

Visual Analytics, Sense-Making, Criminal Intelligence Analysis, Architecture, REST, Security, Ethics, Privacy and Legal (SEPL), Model View Presenter, Isolation, Google Web Toolkit (GWT), Errai, Docker containers, Central Authentication Service (CAS), LDAP, GrayLog, GIT, Gradle, Nexus, Jenkins

I N T E N T I O N A L L Y B L A N K

INTRODUCTION

VALCRI – Visual Analytics for sense-making in Criminal Intelligence Analysis – once completed, will be a TRL 5 laboratory prototype that has been validated in an operational environment. In VALCRI we apply concepts and principles from visual analytics (Thomas & Cook, 2004) to develop a system that facilitates human reasoning and analytic discourse, tightly coupled with semi-automated human-mediated semantic knowledge extraction capabilities.

During the project, VALCRI partners will research and develop a semantic data mining engine informed by a self-evolving crime ontology. This semantic extraction engine will semi-automatically identify interesting crime concepts, extract them, and then make it available to the system so that these concepts may be used by the system to enable further searches based on document similarity or provide the analyst with brief descriptions of what a document or set of similar documents are about. The extracted concepts can also be used by the analysts will also be able to specify or direct further searches. VALCRI will also include an associative search capability to discover unanticipated relationships, or relationships that may exist between data that can only be discovered by combining data from several different data sets. It is intended for the ontology to be updated as new terms or concepts are introduced to the system. In this way the system learns, with the evolving ontology guiding subsequent semantic searches.

VALCRI will also provide functionality to analyse the temporal evolution of criminal networks. Such analytical tools will enable analysts to understand how known relationships and their effect on the community change over time.

In terms of security, data access will be protected by a fine grained access control capability based on both Attribute-based Access Control (ABAC) and Proximity-based Access Control (PBAC). This fine grained access control will enable complex and specific settings to be set to ensure that security requirements are addressed. It is complemented by Secure Logging for transactions, which allows ex-post control of actions, and by Privacy Enhancing Technologies (PET), all of which together are capable of addressing various Security, Ethical, Privacy and Legal (SEPL) requirements.

The aforementioned technologies will be implemented through a configuration management system, OpenPMF, the Open Policy Management Framework (a patented technology developed by partner Object Security Ltd), based on the concepts of Model Driven Security. Using this approach, human-readable high-level policies regarding SEPL (Security, Ethics, Privacy and Legal) requirements are mapped to machine-usable low-level enforcement rules for the different SEPL technologies.

The VALCRI user interface will be operated by tactile reasoning (Takken & Wong, 2015), an interaction technique that supports sense making by the direct manipulation of information objects in the user interface. When one is presented with a set of information that can be freely moved, manipulated, grouped and re-arranged in a visuospatial manner, this interaction method can help us discover meanings or relationships. Such actions are externalisations of our mental processes and have been referred to as epistemic actions (Kirsch & Maglio, 1994). Such a method will be used to support the assembly of evidence to create a case based on argumentation theory and evidential reasoning for the rigorous treatment of data and inferences.

VALCRI will also include functionality for addressing cognitive bias, the use of ethically sensitive data, and complying with legal frameworks, methods and data models for tracking and storing data, process and analytic provenance. Such functionality will be embedded within the system architecture. Where this is not possible, controls and protection mechanisms such as procedures to cope with incidental findings will be incorporated into the socio-organisation structures in which VALCRI will operate.

VALCRI is intended to support a various criminal intelligence analysts' tasks, ranging from strategic analysis functions such as statistical analysis and crime pattern analysis, to the support of individual crime investigations. The system user interface will be designed to support seamless transitions between aggregated views to views showing the data within individual records. This will enable the analyst to work with the lowest level of detail and yet be able to persistently access the situational context. This is important for interpreting data and sense-making.

In the following pages we describe the technical architecture and the technology stack we have constructed to support the functionality briefly described above.

THE SYSTEM ARCHITECTURE

The VALCRI architecture (Figure 1) is built from different Docker containers that speak with each other using mostly REST interfaces. In the diagram below we can see the different Docker containers (blue rounded rectangles). Nested containers are used within some of these containers (e.g. Jetty or ES).

A great deal of attention was spent to incorporating Security, Ethics, Privacy and Legal (SEPL) solutions. SEPL considerations are taken throughout the architecture of VALCRI. As an example, the SEPL Enforcement components control all access to persisted data.

The data that is ingested by the Ingester, or that is provided by the user through the user interface, is kept in the Unstructured Database (UDB) and the Structured Database (SDB). The UDB keeps binary data (e.g. video or audio files). The SDB keeps structured data, for example, police reports which include well defined fields. Note that, unstructured text is usually kept in the SDB as well – for example, a po-

lice report may contain structured data such as address (street name, number, etc.), dates, or information about the person that prepared the report. In addition, it may contain unstructured text, such as a field that contains the description of the event reported.

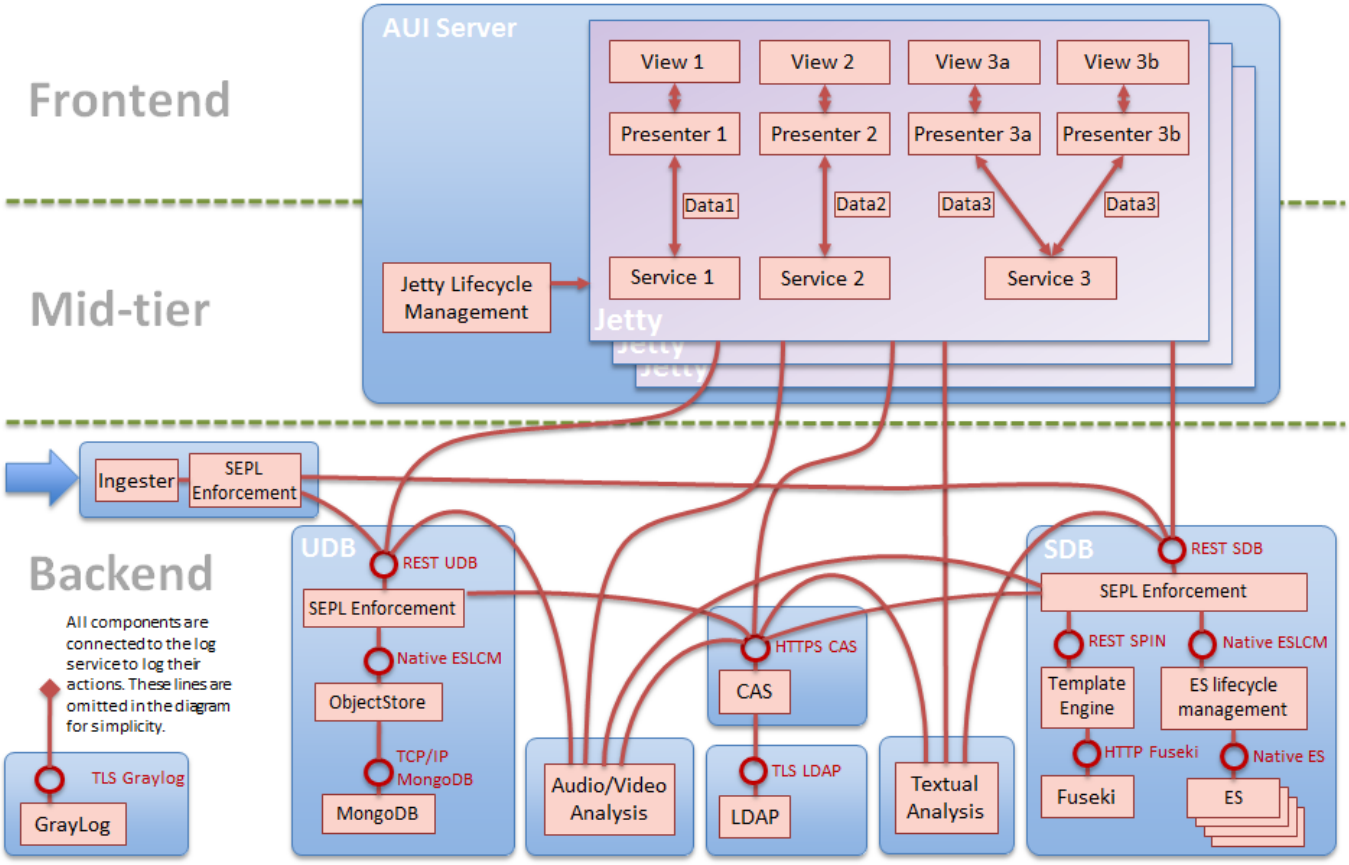

Figure 1. The VALCRI multi-tier system architecture

The SDB involves a SPARQL server - Apache Jena Fuseki, and a multitude of Elasticsearch (ES) instances. Fuski is used for persistency, while Elasticsearch provides quick searches through the data allowing high responsiveness from interactive visualizations.

The SEPL Enforcement component dispatches the queries to Fuseki or to ES. To enforce SEPL, the Template Engine manages the previously checked and accepted query templates that can be sent to Fuseki. SEPL is applied to ES by confining the indexed data for each user. In other words, a specific ES index is created per user to ensure that the user can only explore the data he/she is allowed to see. These ES instances are created by the ES lifecycle management component. To avoid duplicated ES indexes (that is, when a few users are allowed to see the same data), the ES lifecycle management implements the Proxy Design Pattern (Gamma et al, 1994). The queries to ES as well as the output of the ES queries pass through the SEPL Enforcement component, allowing for extra checks if needed.

As said, in the same manner to the SDB, the UDB includes a SEPL Enforcement component. The ObjectStore component is a service for storing binary data. It is integrated with OpenPMF to allow access control which is enforced using policies. These policies are defined using a user-friendly API. The ObjectStore is using and controls a MongoDB instance for persistency.

The Textual Analysis component processes unstructured text. It gets its input from the SDB, and persists its output results back to the SDB. For example, processing an unstructured text description of a crime report might result in the extraction of person names which are then saved back to the SDB as metadata linked to the crime report.

The Audio/Video analysis component gets its input from the UDB, and outputs the results back to the UDB (e.g. anonymized videos) as well as the SDB (e.g. semantics identified in the videos).

The Advanced User Interface (AUI) is a web based user interface. It includes advanced and integrated visual analyt-

ics views that enable the user to better interact and navigate through the data using visualizations.

The AUI server is also designed with SEPL in mind: a Jetty (Java HTTP server and Java Servlet container) instance is created per user by the Jetty Lifecycle Management component. Each such instance lives inside a Docker container to promote isolation. This ensures that a security issue in one of the components running in Jetty, or a security issue in Jetty itself, does not allow a malicious user to access the information of other users.

A Jetty instance hosts the mid-tier services, serves the front-end JavaScript code and manages the communication between the mid-tier services and the front-end. The mid-tier services are written in java and front-end components are implemented using GWT – Google Web Toolkit – to enable front-end Java development. The front-end code is trans-compiled to JavaScript by the GWT compiler. Jetty sends the compiled JavaScript code to the browser running on the user client. The communication between the mid-tier and front-end components is implemented with Errai which abstracts away the communication complexity.

The Model View Presenter (MVP) design pattern (Potel M, 1996) is implemented. The Views are the visible components shown to user in the browser, e.g. a timeline. Each View is managed by a Presenter which provides the necessary logic to present the data in the respected View and to process the user interactions. The Model is the data shared between the Presenter and the Service and used by the communication protocol between both. The transformation logic of getting the data from the backend, and transform it to a Model that can be presented, is done by the respective Service.

Each mid-tier service defines how it needs to communicate with the backend components – the databases: SDB and UDB, and with the analysis components. These services communicate mostly with the user's ES instance. As explained earlier, ES indexes the data from Fuseki and for each user profile another index is available. The mid-tier services can also communicate directly with the SDB or UDB in order to store data that the user inserted or to access specific data that is not available in the ES indexes.

The SEPL Enforcement components (Access Control, Privacy Enhancing Technologies and Secure Logging) in the UDB and SDB communicate to the CAS (Central Authentication Service) component, and they are all governed by a common high-level policy based on the OpenPMF Policy Management Framework, thereby ensuring that only authorized access is granted and that data is appropriately processed before access, thereby addressing various security and privacy goals. In the same manner, the analysis components, and the Jetty Containers, communicate with the CAS component. The user management of the CAS component is linked with the LDAP (Lightweight Directory Access Protocol) component to manage and authenticate user credentials..

All the components may use the interface exposed by the GrayLog component, ensuring that any action done by any component can be logged in the logging storage.

DEVELOPMENT AND TESTING ENVIRONMENT

As a consortium of 18 partners, 10 of which contribute various software components, there is a strong need for a development and testing pipeline that ensures the interoperability and cooperation between each of the components. Therefore, early and systematic system integration is a key principle for the success of VALCRI.

For this purpose, each new component or every change to an existing component in the VALCRI system will pass through a processing chain whereby each stage deals with a different integration aspect of the component in question. This processing chain of is more commonly referred to as the development and testing pipeline.

This pipeline aims to provide continuous system integration while promoting collaboration, contributions, quick feedback to contributions, changing and evolving interfaces, and above all respecting the principle of "keep it working" – allowing to introduce many contributions in small steps while the system continues to compile and work.

Note that there is a subtle trade-off between the rigour and flexibility of the development process. On the one hand, the development and testing pipeline must be thorough in order to enforce system stability. But, on the other hand, when the pipeline is too strict, it might slow down quick changes or additions.

Figure 2 depicts the four processing stages from which the pipeline is built up from. Each stage takes care of a different aspect of the process. It also shows how a developer can interact with the pipeline, and which stages are triggered in response.

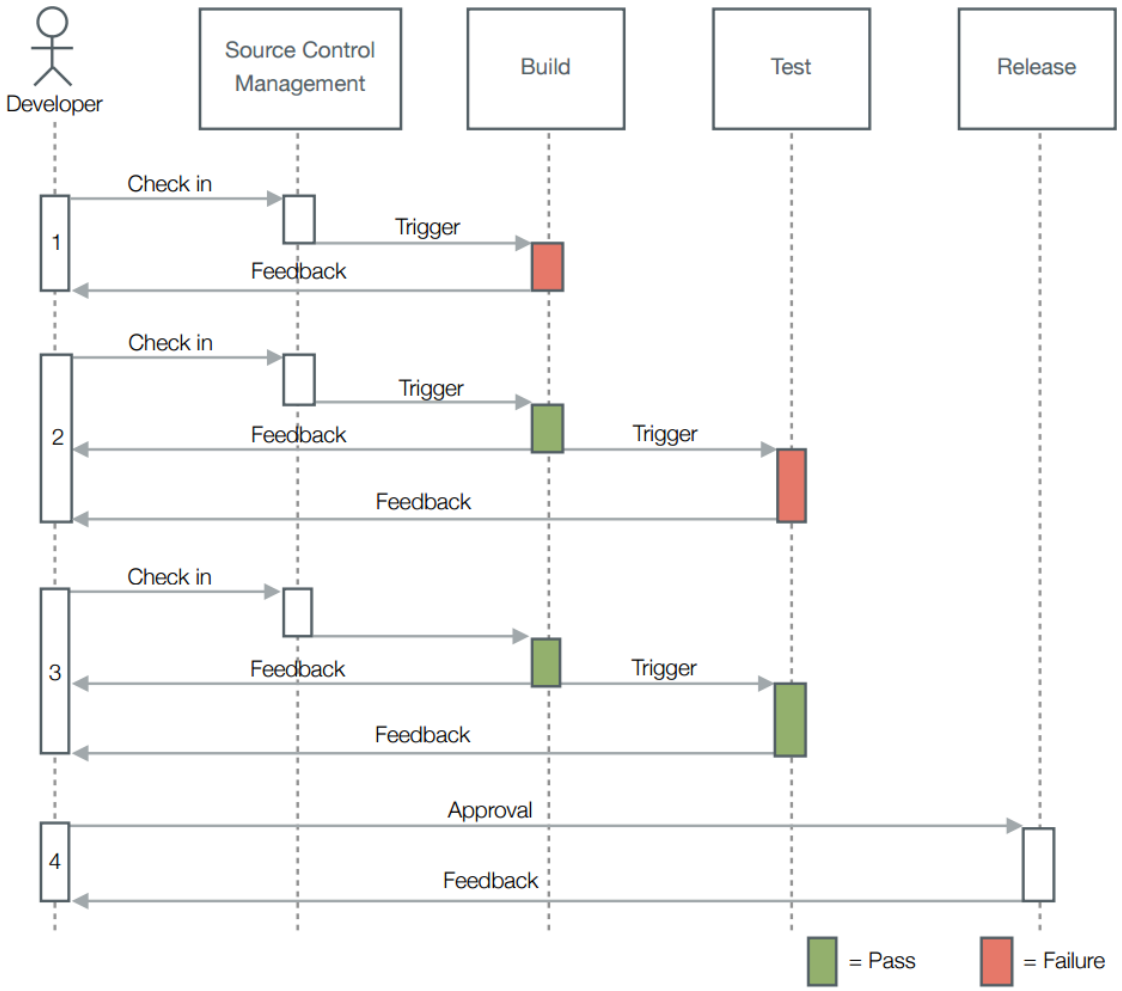

Figure 2. The Four processing stages of the development pipeline

The GIT Source Control Management (SCM) (https://git-scm.com/) platform is used for keeping track of changes in the source code. It is accompanied by a GitLab (https://about.gitlab.com/) installation that provides a user interface that allows managing the user accounts and the individual code repositories. The choice of using GIT seemed to be the most natural fit as it features for decentralising the code, smooth handling of branches and seamless integration with other development tools necessary for VALCRI software development.

In order to compile and build the code available from the SCM, a custom build system was developed using Gradle (https://gradle.org/). Gradle provides a Groovy (http://www.groovy-lang.org/) Domain Specific Language (DSL) and is used to define and automate all the details for managing and building the VALCRI system. The Gradle set-up is also accompanied with a Nexus (http://www.sonatype.org/nexus/) component or artefact repository which hosts all the compiled, binary VALCRI system components. Together, Gradle and Nexus need to perform the bulk of the tasks in the development and testing pipeline. Therefore, they have many requirements to adhere to. The following sections will iterate through each of those requirements and discuss how they were fulfilled.

Gradle

Gradle has a number of useful and important features:

Transitive dependency resolution — Gradle can resolve dependencies from Maven and Ivy artefact repositories. It also automatically traverses down the dependency tree to fetch the dependencies and the dependencies of the dependencies.

Support for Nexus — Nexus is developed by Sonatype, the company behind Maven. Hence, Nexus is a fully Maven-compliant artefact repository and is thus well supported by Gradle.

Support for build promotion — we have designed the Gradle build system in a way that developers only need to provide a component id and version. Gradle can then automatically upload the component to Nexus.

Allow for maintenance — this is one the key strengths of Gradle as it is based on a JVM programming language

named Groovy. As a result, maintaining the build system is like any other software component written in Groovy.

Multiple platform support — besides the requirements in D2.3, it is also noteworthy to mention the fact that Gradle is completely Java based. This means that it can operate on any OS as long as the VALCRI developer has installed a suitable JVM.

IDE support — from the very beginning of the project, all technical members of the consortium agreed on using Eclipse as the default IDE. Gradle automatically generates the configuration files needed for importing the projects into Eclipse. However, it is also possible to use other IDE's if preferred.

Nexus

Nexus has the following useful features:

Allow continuous collaboration — using Gradle, a developer can checkout or create a new software component for the VALCRI system and integrate it within the large multi-component hierarchy without having to deal with the entire source tree.

Streamlining sharing of artefacts — developers can use Gradle to upload a compiled VALCRI software components to Nexus in order to share it with others.

Privacy and security — each partner can choose which components they want to share to whom. Furthermore, each partner has received a personal account to manage their own artefact repository in Nexus.

Speed — the Nexus code repository hosts all the precompiled VALCRI components. VALCRI developers can fetch these instead of having to recompile all of the VALCRI system components themselves. This drastically speeds up the build process.

Reliability — release artefacts can only be uploaded once to the repository, so they are guaranteed to stay the same over time. If a developer wants to change a component, he has to increment its version number and upload a new version to Nexus. This ensures that the Gradle can still resolve the previous versions and reliably build the VALCRI system. It is the responsibility of the developer to then make sure that all components using the old version are upgraded to use the new version. For better development support, Nexus also allows to publish snapshot artefacts without the need to increment version numbers. Each change to a library in development can be pushed as a new snapshot and consumed like a normal artefact. The only restriction to snapshots is that they can never be part of a final release of the VALCRI system.

Backup — all artefacts uploaded to the Nexus repositories are automatically backed up.

Integration with build tools — As discussed above, Gradle supports Nexus out of the box.

Multiple component types — Nexus is primarily focused on Java artefacts (JAR files). This conforms to the overall focus of the consortium. It also supports plain ZIP files which can basically contain anything.

For each change in the SCM, the source code is automatically rebuilt and all tests are run. In order to do this, a Jenkins (https://jenkins-ci.org/) installation was put into place. Jenkins continuously polls for changes in the SCM. Upon each change, it runs a build and test cycle over the full code base. Upon any failure, it will send an email to the VALCRI developer mailing list informing all technical partners about the failure. JUnit (http://junit.org/) was selected as the means to define unit tests and Selenium to run integration tests.

Docker

In VALCRI we have chosen Docker (https://www.docker.com/) as the primary tool to solve this problem. As nicely described on their website, Docker is a platform for building, shipping and running the complete environment for distributed applications. It allows you to package your application and all its run-time dependencies into a standardized unit of deployment: a container. Each container can thus wrap a VALCRI software component in a complete filesystem that contains everything it needs to run: the executable binary, system tools, system libraries - anything you can install and execute on a server.

Getting the VALCRI system up and running while provisioning it with the correct configuration and environment is very complex. Even for small applications this can quickly become complicated and time consuming as each application has its own unique set of run-time dependencies and configuration files. These run-time dependencies are different from the compile-time dependencies that are resolved by the build system. In essence, the dependencies of the build system are comprised of the internal VALCRI software components depending on each other and the software libraries they are built from. Even though the development and testing pipeline (refer to D2.3 version 1) is able to weave together all those compile-time dependencies into one deployable and executable unit, this binary still has run-time dependencies to an operating system, networking and hardware facilities, logging, identity management, etc. These dependencies are usually off-the-shelf services that are managed outside VALCRI. Yet, they need to be available and configured properly in order for the VALCRI system to work and remain secure.

In this context, the term infrastructure represents all supported environments together with the services that support the VALCRI application. The process that prepares the environment for deployment is the main focus of this section. The tool that implements this process for VALCRI should adhere to the following requirements:

Versioned — the state of the infrastructure should be specified in a configuration file that is versioned in the Source Control Management (SCM).

Automatic — reestablishing a known good environment configuration must be simple and fast. Therefore, run-time dependencies needed to run the VALCRI components and setup the complete run-time environment should be automatically fetched. This is also a necessary condition for automating the process of integration, testing and deployment.

Efficient — preparing the deployment environments should be fast or at least a one-time cost. This allows developers to keep the time between changing the code and running the updated executable in their environment as short as possible. Also, the tool should not induce a significant performance overhead when running the VALCRI components. The ultimate goal is to allow VALCRI developers to employ commodity hardware for most scenarios.

Modular — As the VALCRI architecture is comprised of many different components, the tool should support lifting those out and run and debug them in their own custom environment. This also gives unique opportunities in terms of security as the individual VALCRI software components are then isolated from each other so one can better control the data flow between them.

Interoperability — the tool should be able to run on both Windows and Unix-based platforms.

CONCLUSION

Several challenges that are derived by the project targets are tackled by the architecture. Visual analytic user interfaces require the user to interact with the visualizations and therefore the underlying analytics must happen with minimal delays. On the other hand, complex constraints are put into place in order to ensure that the SEPL issues are well addressed. In support of this, we employed a software architecture using a combination of the latest development approaches for advanced user interfaces and container-based software.

The VALCRI project challenges the development process due to its high complexity, large size and large consortium of academic and industry partners spread around Europe. This challenge was met by the careful design and implementation of a development and testing pipeline.

REFERENCES

Gamma E., Vlissides J., Johnson R., Helm R. (1994). Design Patterns: Elements of Reusable Object-Oriented Software.

Kirsch, D., & Maglio, P. (1994). On distinguishing epistemic from pragmatic action. *Cognitive Science, 18*(4), 513-549.

Potel M (1996). MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. Taligent, Inc.

Takken, S., & Wong, B. L. W. (2015). Tactile reasoning: Hands-on vs. Hands-off - what's the difference? *Cognition, Technology & Work, 17*(3), 381-390. doi:10.1007/s10111-015-0331-5

Thomas, J. J., & Cook, K. (Eds.). (2004). *Illuminating the path: A research and development agenda for Visual Analytics*: IEEE CS Press.

The research leading to the results reported here has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) through Project VALCRI, European Commission Grant Agreement Number FP7-IP-608142, awarded to Middlesex University and partners.

	VALCRI Partners	**Country**
1	Middlesex University London Professor B.L. William Wong, Project Coordinator Professor Ifan Shepherd, Deputy Project Coordinator	United Kingdom
2	Space Applications Services NV Mr Rani Pinchuck	Belgium
3	Universitat Konstanz Professor Daniel Keim	Germany
4	Linkopings Universitet Professor Henrik Eriksson	Sweden
5	City University of London Professor Jason Dykes	United Kingdom
6	Katholieke Universiteit Leuven Professor Frank Verbruggen	Belgium
7	A E Solutions (BI) Limited Dr Rick Adderley	United Kingdom
8	Technische Universitaet Graz Professor Dietrich Albert	Austria
9	Fraunhofer-Gesellschaft Zur Foerderung Der Angewandten Forschung E.V. Mr. Patrick Aichroft	Germany
10	Technische Universitaet Wien Assoc. Prof. Margit Pohl	Austria
11	ObjectSecurity Ltd Mr Rudolf Schriener	United Kingdom
12	Unabhaengiges Landeszentrum fuer Datenschutz Dr Marit Hansen	Germany
13	i-Intelligence Mr Chris Pallaris	Switzerland
14	Exipple Studio SL Mr German Leon	Spain
15	Lokale Politie Antwerpen	Belgium
16	Belgian Federal Police	Belgium
17	West Midlands Police	United Kingdom