SmartCLIDE

"The Stairway to Cloud"

This project has received funding from the European Union's Horizon 2020 research and Innovation programme under grant agreement No 871177

Content

CONTENT 1
Figures.................... 2
Tables.................... 2
PROJECT CONSORTIUM.................... 3
INTRODUCTION.................... 4
Context and motivation.................... 4
Challenge.................... 4
Solution.................... 5
Impact.................... 6
Benefits for the targeted users.................... 6
LET'S LAY THE FOUNDATION.................... 8
Cloud Computing in a nutshell.................... 9
Machine Learning and Deep Learning: A power couple.................... 13
SmartCLIDE: a new cloud-native IDE.................... 18
Service Discovery in a Nutshell.................... 20
Programming By Example.................... 26
OUR SCENARIOS OF USE.................... 29
About LoRaWAN communication services.................... 30
Enhance IoT-Catalogue with an integrated Cloud IDE.................... 34
Provide a Quick Demonstration for a Customer.................... 36
Optimizing Resources.................... 40
DEEP DIVE.................... 42
SmartCLIDE Innovative Approaches.................... 43
SmartCLIDE Market Requirements.................... 47
SmartCLIDE Service Creation.................... 54
SmartCLIDE Deep Learning Engine.................... 57
SmartCLIDE User Interface.................... 60
SmartCLIDE DLE Component.................... 66
BACKEND SERVICES.................... 69
Source Code Repository.................... 70
Service Discovery, Creation and Monitoring.................... 72
Security.................... 77
Intercommunication.................... 80
User Access Management.................... 82
Deployment and CI/CD.................... 83
Tool Support for Architectural Pattern Selection in Cloud centric Service-oriented IDEs.................... 86
Runtime Monitoring and Verification (RMV).................... 89
Vulnerability prediction based on Text Mining and BERT.................... 93
Testing Cloud-Based Applications.................... 98

Figures

Figure 1: Pizza Analogy.......... 10
Figure 2: Perceptron.......... 15
Figure 3: Deep Learning Engine and Services.......... 16
Figure 4: The Self-Registration Pattern.......... 21
Figure 5: The Third-Party Registration Pattern.......... 22
Figure 6: A Service Description.......... 22
Figure 7: Client-Side Discovery.......... 23
Figure 8: Server-Side Discovery.......... 23
Figure 9: A low-power wide-area network (LPWAN).......... 30
Figure 10: Monolithic vs Microservice architecture.......... 31
Figure 11: Deploying an application to a Kubernetes cluster.......... 32
Figure 12: : Real-time communication platform system.......... 32
Figure 13: IoT Catalogue.......... 34
Figure 14: SmartCLIDE research problems identification.......... 44
Figure 15: Monolithic vs. Microservices architecture.......... 49
Figure 16: Service Creation Widget.......... 54
Figure 17: Technical Debt Principal widget.......... 55
Figure 18: Technical Debt Interest widget (per File and Evolution).......... 56
Figure 19: BPMN Editor.......... 60
Figure 20: Eclipse Theia code editor.......... 61
Figure 21: Security analysis page.......... 61
Figure 22: Vulnerability assessment page.......... 62
Figure 23: The main page of the services.......... 62
Figure 24: Code auto-completion.......... 63
Figure 25: Live template recommendation.......... 63
Figure 26: Comments generation.......... 64
Figure 27: Main page of the deployments.......... 64
Figure 28: Main page of the cost comparison service.......... 65
Figure 29: Runtime metrics monitoring and visualization page.......... 65
Figure 30: Hierarchical Group Structure in GitLab.......... 67
Figure 31: Runtime Monitoring and Verification Component Diagram.......... 71
Figure 32: Software Security Assurance Module.......... 74
Figure 33: MOM Component Diagram.......... 78
Figure 34: Set of Applications Diagram. Workflow.......... 81
Figure 35: CI Server & Testing and QA Component Diagram.......... 83
Figure 36: RMV Subsystem Overview.......... 85
Figure 37: Transformer encoder.......... 91
Figure 38: SmartCLIDE External Service Identification.......... 95

Tables

Table 1: Micro-planning per task and relevant problems.......... 44
Table 2: Popular Related Pre-trained AI models.......... 68

PROJECT CONSORTIUM

INTRODUCTION

The SmartCLIDE project enables organizations on the path to digitalization to accelerate the creation and adoption of Cloud and Big Data solutions. The innovative smart cloud-native development environment will support creators of cloud services in the discovery, creation, composition, testing, and deployment of full-stack data-centered services and applications in the cloud.

Context and Motivation

The SmartCLIDE research project aims to bridge the gap between on-demand business strategies and the lack of qualified software professionals by creating a new cloud native IDE that makes it easier to develop and deploy cloud services. The project is funded by the European Union's Horizon 2020 research and innovation program, and involves a consortium of 11 partners from Germany, Greece, Luxembourg, Portugal, Spain, and the United Kingdom.

SmartCLIDE extends Eclipse Theia to provide a development environment that makes it easy to create, compose, test and deploy data-centric full-stack services and applications in the cloud. In addition to providing high levels of abstraction at all stages (development, testing, deployment and execution), SmartCLIDE makes it easy for IaaS and SaaS service self-discovery. The project covers the architecture, front-end and back-end services of the cloud based IDE.

Challenge

In this context, when companies face the creation or composition of new services for their clouds, they are having three alternatives, each one being subject to different problems/limitations:

1. Development of services from scratch enclose a high complexity due to the wide variety of technologies that shall be used in the whole stack. It is expensive and time consuming.
2. Creating new services by composition: Existing marketplaces are tightly coupled to IaaS and PaaS providers, and they are not always uniformly classified or well documented, so the discovery of valuable and secure services are mostly a manual

process and its validity is demonstrated by trial and error.

3. Pricing models of public cloud providers are very complex since they combine different variables depending on the type of service. These variable can be time of usage, resources used (memory, storage, processing capacity), thousands of predictions obtained (in the case of machine learning algorithms), volume of data transferred and many more. This fact makes the calculation of costs extremely difficult to predict, and therefore to control.

Solution

Eclipse OpenSmartCLIDE project originated from the SmartCLIDE project. The concept for the IDE and the architecture are detailed in this document. All services developed within SmartCLIDE are open-source and are licensed under the Eclipse Public License 2.0 scheme.

OpenSmartCLIDE is based on Eclipse Theia, which provides all of the tools necessary for development. Theia consists of a rich interface with a vast range of features that accelerate deployment of cloud services, improve their quality, and expand the skills of novice and experienced developers.

The main features of OpenSmartCLIDE include:

1. **Life cycle support.** Software follows a life cycle, from feature specification to solution deployment. OpenSmartCLIDE provides the specific tools required at each life-cycle stage. For example, at the development stage, OpenSmartCLIDE provides data sources, data transformations, graphics visualization artefacts, and general-purpose abstractions and patterns that can be combined to implement features.

2. **Insightful source code monitoring.** OpenSmartCLIDE includes visualization features that help developers gain deeper understanding of the source code. It dynamically shows the meaning of expressions or code flow at low levels of granularity. It also allows developers to compare different software states, perform state changes that are reflected dynamically, and create new abstractions that can be easily reused.

3. **CI/CD integration.** OpenSmartCLIDE enables integration with widely used CI/CD tools such as GitHub and GitLab. The Eclipse OpenSmartCLIDE also includes innovative features that leverage the power of a deep learning engine:

4. **Development by demonstration and text notation.** OpenSmartCLIDE automatically retrieves resources that are considered relevant for the new development. The end user can use text notation to enhance the description of the retrieved behaviour or algorithm. The deep learning engine then uses these notations to suggest programmatic solutions that result in

the desired output.

5. **Automatic software classification.** The deep learning engine automatically identifies and classifies existing and new software abstractions that can be visualized in the IDE for reuse based on the purpose or behaviour defined by the end user.

Impact

We can list four major impacts resulting from this research project:

- **Impact 1:** Contribute to the development of an ecosystem that will respond to the future digitisation needs of industry and the public sector. SmartCLIDE IDE provides the baseline for the establishment of an ecosystem of cloud service creators that will be able to share services and applications that can be automatically deployed in the cloud.
- **Impact 2:** Assist the development of new cloud-based services and infrastructures in Europe and foster an industrial capability in the cloud computing sector. The disruptive technology proposed by SmartCLIDE based on the coding-by-demonstration principle, will allow users with low technical skills to create and securely deploy data intensive services of the highest quality.
- **Impact 3:** Create new opportunities to encourage European-based providers, in particular SMEs, to develop and offer cloud-based services based on the most advanced technologies. SmartCLIDE proposes the utilization of existing open-source code to create the baseline upon which the new IDE will be developed, optimizing the use of technological resources and the need of investments to further develop the solution, facilitating the access of SMEs to the technology.
- **Impact 4:** Leverage research and innovation projects to support the development and deployment of innovative cloud-based services and next generation applications, for the public and private sectors (including standardisation and applications for Big-Data and other sector-specific applications).

Benefits for the targeted users

OpenSmartCLIDE introduces several benefits for the different stakeholders within the service creation lifecycle:

It lowers the entry-level to programming activities to non-technical staff

SmartCLIDE allows end-users to easily prototype features (that can be enhanced later on by developers). It also provides a powerful training tool for novel developers to understand the underlying mechanisms of data-intensive applications

It increases the reusability of services

SmartCLIDE allows reusing existing and ad-hoc created microservices, data, control structures, or operations abstractions.

It improves of the transparency, readability, and comprehensibility of software

SmartCLIDE will implement several features that will improve the readability and comprehensibility of software

- it implements the coding-by-demonstration principle, which is a way of creating software that is closer to the way humans think. Rather than adapting the users' mental scheme to the requirements of a programminglanguage, it's the user who instructs a system to reach the desiredresult making use of abstractions.
- it implements control flow monitoring at run-time (even at low levels of granularity), which improves the knowledge about how the software works and interacts with different components and sub-systems.
- it implements a stateful behavior, showing the state of data (and variables) at each execution step.

It increases quality and security levels

At design time, when the user defines the output of the program, she will make use of domain-specific languages like Gherkin, which will enable the full automation of acceptance tests making use of natural language. SmartCLIDE will also make available full abstractions of other testing frameworks at testing stage, enabling coders to deploy containers with testing frameworks for testing purposes in seconds.

It increases productivity levels

- Improved reusability, higher comprehension of the underlying mechanisms of soft-ware, and full control over the lifecycle (from specification to deployment) will boost the productivity of development teams, even in the most complex contexts.

LET’S LAY THE FOUNDATION

This first set of articles presents the pillars of our project: Cloud Computing, Deep Learning, the Integrated Development Environment, Service Discovery and Programming by Example.

Our partners have made a special effort to write for as broad a technical audience as possible, to provide a look into the state-of-the-art of the project pillars and to understand the innovations that the SmartCLIDE project plans to implement.

Cloud Computing in a nutshell

by Netcompany-Intrasoft

Cloud computing has become the platform for the new, global digital transformation stage we have entered to not only for our countries, governments and companies but also for each one of us. Our phone contacts, photos and messages are stored in cloud data centers. Music and videos are being delivered through high capacity cloud streaming services. The best route finding filter on maps with live traffic information is made possible with Artificial Intelligence cloud services. Tax income declaration is applied through the cloud. Even, registration to kindergarten schools in Greece is being performed for the first time by using online family status verification through the public registry, again by leveraging cloud technologies! Cloud computing is ubiquitous; either we use it deliberately or we do not.

But what is the actual meaning of this "cloudy" term? Is data being transferred to the sky clouds for some purpose? Probably not!

Cloud Computing is the on-demand delivery of computing services such as servers, databases, networking structures and software over the internet. This is implemented by dedicated data centers and server farms whose services are available to many different customers/users, offering faster innovation, flexible resources, and economies of scale. Cloud computing services are based on a "pay-as-you-go" model which means that clients are only charged for the services they use.

Cloud computing centers are divided into three major categories.

- **Public clouds** are operated by third-party companies such as Microsoft (Azure) or Amazon (AWS) which are responsible for the stability, maintenance and expansion of the underlying infrastructure, and provide their service over a public network, the internet.
- **Private cloud** is structurally identical to the public cloud, but it is being owned and used by a single organization while the provided services are restricted to a private network.
- **Hybrid cloud** combines private and public clouds connected with technology that enables data and applications to be interchanged. This interconnection offers higher flexibility and more deployment options.

Types of Cloud Computing services

Cloud Computing can be provided through different models according to the abstraction level and the complexity of the underlying services. The three standard models according to NIST are:

- INFRASTRUCTURE AS A SERVICE (IAAS): The basic category of Cloud Computing. Users rent servers, networking hardware, storage devices and operating systems and configure them to provide business value. Users are responsible for system configuration, operating system updates and vulnerability eliminations.
- PLATFORM AS A SERVICE (PAAS): Ideal for developers who want to quickly deploy a web application without having to setup servers, operating systems and networking, as they are already configured. In other words, PaaS is an environment created on-demand for developing, delivering and administering web applications.
- SOFTWARE AS A SERVICE (SAAS): Here the whole application lifecycle, as well as the underlying infrastructure, the configuration and administration tasks, are performed by the cloud provider. Users interact with the application through a web browser or mobile device.

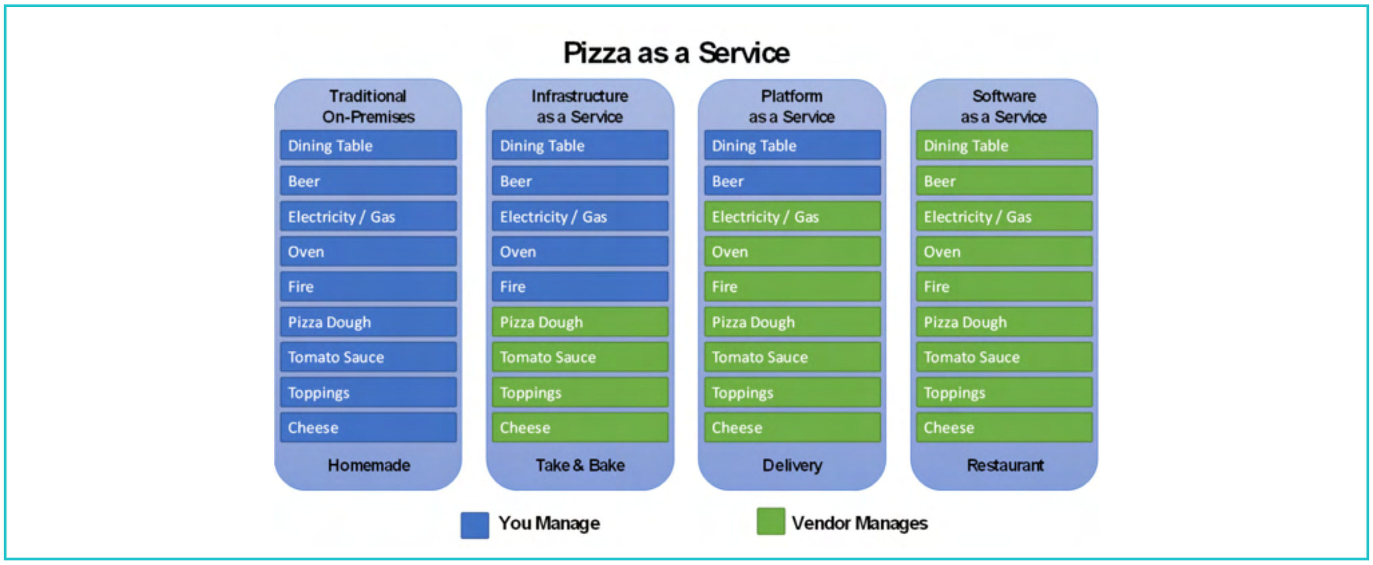
The Pizza Analogy

All these new terms can be confusing. Even experienced software engineers found the concept delineation difficult. For this purpose, the famous Pizza Analogy has been created. In the first column, there is the equivalent to the non-cloud traditional application deployment process. All tasks are being handled by the user. While on the other hand, the last column represents the equivalent to the SaaS approach where every single task is being "outsourced" to the Cloud Service Provider leaving the user only with the pizza delight!

Latest advances

Cloud-Native

Inevitably, the transition to Cloud Computing was not spontaneous. Cloud services were initially



Pizza as a Service

Traditional On-Premises	Infrastructure as a Service	Platform as a Service	Software as a Service
🟦 Dining Table	🟦 Dining Table	🟦 Dining Table	🟩 Dining Table
🟦 Beer	🟦 Beer	🟦 Beer	🟩 Beer
🟦 Electricity / Gas	🟦 Electricity / Gas	🟩 Electricity / Gas	🟩 Electricity / Gas
🟦 Oven	🟦 Oven	🟩 Oven	🟩 Oven
🟦 Fire	🟦 Fire	🟩 Fire	🟩 Fire
🟦 Pizza Dough	🟩 Pizza Dough	🟩 Pizza Dough	🟩 Pizza Dough
🟦 Tomato Sauce	🟩 Tomato Sauce	🟩 Tomato Sauce	🟩 Tomato Sauce
🟦 Toppings	🟩 Toppings	🟩 Toppings	🟩 Toppings
🟦 Cheese	🟩 Cheese	🟩 Cheese	🟩 Cheese
Homemade	Take & Bake	Delivery	Restaurant

🟦 You Manage 🟩 Vendor Manages

Figure 1 : Pizza Anology

used mainly as an infrastructure (IaaS), that is online, on-demand Virtual Machines which hosted operating systems configured by the end-user. As cloud services were developing, it became obvious that the effective leverage of cloud advantages could bemade possible only by applications tailor-made forthe cloud environment, or as they became known, Cloud Native. Applications of this typeare especially designed and developed for cloud deployment. They are built on microservices architectures, leverage scaling features, and benefit from continuous delivery to achieve reliability and rapid response to the requirements imposed by business changes.

Multi-cloud

Multicloud is the employment of cloud services from different service providers in a single heterogeneous architecture to meet different technical or business requirements. Usually, it is implemented by distributing cloud-native applications to several cloud-hosting environments. The main reasons that favor multi-cloud deployments include reducing reliance on a single vendor, increasing flexibility and adhering to local data protection policies.

Benefits and Pitfalls

The advantages of Cloud Computing can be easily identified. A third-party company that specializes in server hosting and deployment can achieve economies of scale and provide safer infrastructure that is already updated with the latest vulnerability updates. It also guarantees a reliable platform with zero downtime and the most important, offers global-scale availability. This enables a small startup somewhere in the world to deploy a new innovative web application with global availability by paying only for the computing power, traffic and services it uses. Only consider the costs of these requirements for a self-hosted infrastructure alternative!

Top Cloud Providers according to revenue

1. Amazon Web Services
2. Microsoft Azure
3. Google Cloud
4. Alibaba Cloud
5. IBM Cloud
6. VMWare Cloud
7. Hewlett Packard Enterprise
8. Cisco Systems
9. Salesforce
10. Oracle Cloud

Bibliography

Barron, Albert. “Pizza as a Service.” Accessed May 31, 2020.

https://www.linkedin.com/pulse/20140730172610-9679881-pizza-as-a-service/

“Cloud Computing.” In Wikipedia, May 30, 2020.

https://en.wikipedia.org/w/index.php?title=Cloud_computing&oldid=959841571

“Cloud Computing @ Microsoft Azure.” Accessed May 31, 2020.

https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/

“Cloud-Native Applications | Microsoft Azure.” Accessed May 31, 2020.

https://azure.microsoft.com/en-us/overview/cloudnative/

Drake, Nate, Brian Turner December 20, and 2019. “Best Cloud Computing Services of 2020: For Digital Transformation.” TechRadar. Accessed May 31, 2020.

https://www.techradar.com/best/best-cloud-computing-services

McLellan, Charles. “Multicloud: Everything You Need to Know about the Biggest Trend in Cloud Computing.” ZDNet. Accessed May 31, 2020. https://www.zdnet.com/article/multicloud-everything-you-need-to-know-about-the-biggest-trend-in-cloud-computingMell, Peter, and Tim Grance. “The NIST Definition of Cloud Computing.” National Institute of Standards and Technology, September 28, 2011.

https://doi.org/10.6028/NIST.SP.800-145

“Multicloud.” In Wikipedia, February 15, 2020.

https://en.wikipedia.org/w/index.php?title=Multicloud&oldid=940901545.Cisco

“What Is Cloud Computing? – Cloud Computing Definition.” Accessed May 31, 2020.

https://www.cisco.com/c/en/us/solutions/cloud/what-is-cloud-computing.html

Machine Learning and Deep Learning: A power couple

By AIR Institute

Buzzwords like Machine Learning and Deep Learning have been around for quite some time. We've always known that intelligent systems had been a promising technology that would enable us to search through vast amounts of information quickly and effectively, facilitating the discovery and application of knowledge. Over the last decades, technology has successfully taken the reins of numerous tasks that require different degrees of intelligence. In fact, many of the services offered by today's biggest companies are based on Artificial Intelligence, such as Apple's Siri or Amazon and Netflix's recommendation engines.

However, it's important that we learn to distinguish between the two technologies; Deep Learning (DL) and Machine Learning (ML) are two different concepts.

Machine Learning is a branch of computing; it is a very extensive subfield that aims to provide computers with "intelligence". It strives to develop the machine's ability to learn so that it can find the correct solution to a problem without any further explicit programming. Thus, in Machine Learning, systems learn to solve puzzles by themselves.

Researchers in the field of Machine Learning are concerned with developing mathematical approaches and determining the parameters that can be used to solve different problems. At present, we can choose from a range of Machine Learning algorithms, such as classification, regression, dimensionality reduction or clustering. Our choice will of course depend on the type of problem being dealt with. In short:

- **Classification algorithms** assign categories to unseen samples
- **Regression algorithms** predict numeric values from samples
- **Dimensionality reduction algorithms** search for alternative mathematical representations of the data
- **Clustering algorithms** group samples depending on their similarity

Both Machine Learning and Deep Learning have experienced some ups and downs along the way. At times, ML was ahead of DL in terms of interest and at others, DL was ahead of ML. It all depended on the computing performance and their ability to match expectations at a given point in time.

Modeling algorithms

In ML, the term “intelligence” refers to a specific type of intelligence. Unlike an all-purpose, general AI, ML intelligence enables a system to provide a degree of assistance to the user; a helping hand that supplements the human skills or wisdom with knowledge automatically extracted from datasets via mathematical and computational techniques. This implies knowledge is obtained not through programming, but through “training”.

To this end, a model is built so that the system can make predictions on the basis of an input dataset, being the mathematics behind the model that drives the entire learning process. This learning process usually involves adjusting weights -called parameters- during the training phase to ensure that predictions are valid in terms of accuracy, mean error, or inertia, depending on the nature of the data and algorithm. A fine-tune with statistical quality enhancement purposes is performed by finding values in a non-automatic way for the so- called hyperparameters. These numbers have to be specified by the user in a predefined grid.

Models are then successively and iteratively defined, trained, evaluated and tested with different portions of the data to make parameter adjustments. Overfitting is to be avoided here: a phenomenon by which a model would stick too much to the training data, returning biased predictions, making it less general and thus less useful. Likewise, the information contained in the dataset will have to be pre-processed to ensure the desired degree of accuracy and interpretability.

If learning is supervised, the algorithm will compare the results with tagged data (this process requires manual tagging of all the data, which is costly, cumbersome and virtually impossible in **BIG DATA** terms), helping to determine if the model was right or wrong for every sample. On the contrary, if no tagging data is available, we’ll stick to different unsupervised learning algorithms like those for clustering, feature extraction and dimensionality reduction to extract information from our datasets.

NEURAL NETWORKS can roughly be considered a subset of these Machine Learning techniques. They are particularly useful when it comes to problems related to unsupervised datasets or Big Data, making it possible to automatically extract valuable information from patterns.

Approaching Neural Networks: Deep Learning

Deep Learning itself extends Machine Learning, focusing on Big Data and GPU processing -not necessary but convenient. Neither of them is a one-size-fits-all tool for all problems.

Software neurons are simple processing units which simulate -to some extent- the work of their biological counterpart. A neuron has some weighted inputs and an output, to which an activation function is applied. They are grouped in layers, linking one’s outputs with the following inputs – there are different variants of this structure. A layer can contain an undetermined number of neurons.

Neural networks are composed of a number of combinations of layers, each one performing different simple operations which make up a complex "reasoning" process when combined. These layers fall into three categories: input, output, and hidden (the ones in between). Optimization functions are applied to infer the adequate weights for each neuron; hence the computation-demanding nature of these processes.

Taking the widely used example of a handwritten number or an image classification problem, each of the layers would be responsible for identifying details as a border, a particular shape pattern, or performing any of the former with a specific degree of accuracy. To put things in perspective, this can also be done by Machine Learning algorithms, such as Support Vectorial Machine(SVM), by a different implementation approach.

The Deep Learning concept refers to training Neural Networks with more than two hidden layers, independently of how deep the Neural Network is.

A vast variety of Neural Network configurations is available nowadays, but these are the most popular ones:

Perceptron

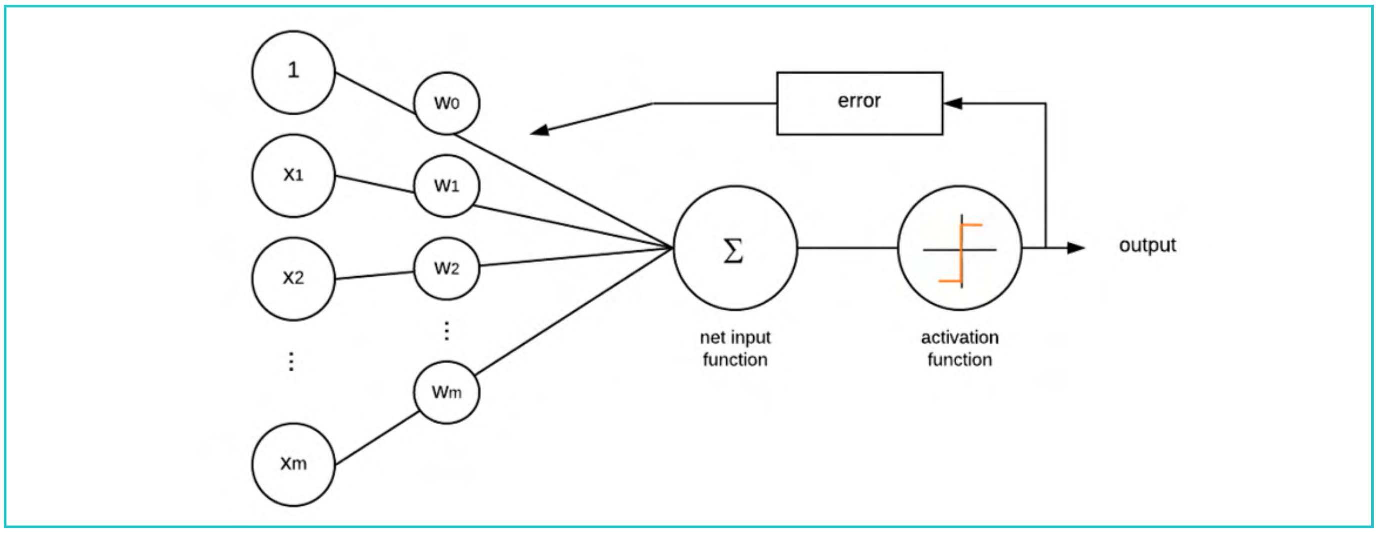


Figure 2: Perceptron

This is the simplest model, in which neurons apply the activation function over the weighted inputs and turn it directly into an output. A multi-layer (one hidden layer + input layer + output layer) perceptron-composed version called Vanilla Neural Network enhances this behavior by adding a layer of heavily interconnected neurons. This is made possible by a backpropagation algorithm, which allows to calculate the loss of a neural network or, in other words, a function that has to be minimized to enhance the quality of the predictions.

Convolutional Neural Networks

These Neural Networks take an image as an input and return another as an output. A common example is object identification in images. They decompose the problem into simpler ones by applying filters to the original channel-decomposed(RGB) information. Recent applications in the field of malicious code identification have had impressive results.

Recurrent Neural Networks

These ones are focused on the identification of patterns in sequences of data. Some of them are given a small amount of "memory", being neurons capable of remembering prior states through a "thinking" process (Long/Short Term Memory LSTM). The most popular application of this type of Neural Network is Natural Language Processing.

Several other groups are to be mentioned, such as Recursive Neural Networks -image treatment and NLP- or Unsupervised Pretrained Networks -data generation and unsupervised learning-.

A range of tools can be used to develop Neural Networks. Luckily enough, some are at a quite high level of abstraction, such as the widespread Keras or PyTorch. Other, lower-level tools include Google's popular Tensorflow which offers Graphics Processing Unit capabilities.

How can Deep Learning and Machine Learning help SmartCLIDE

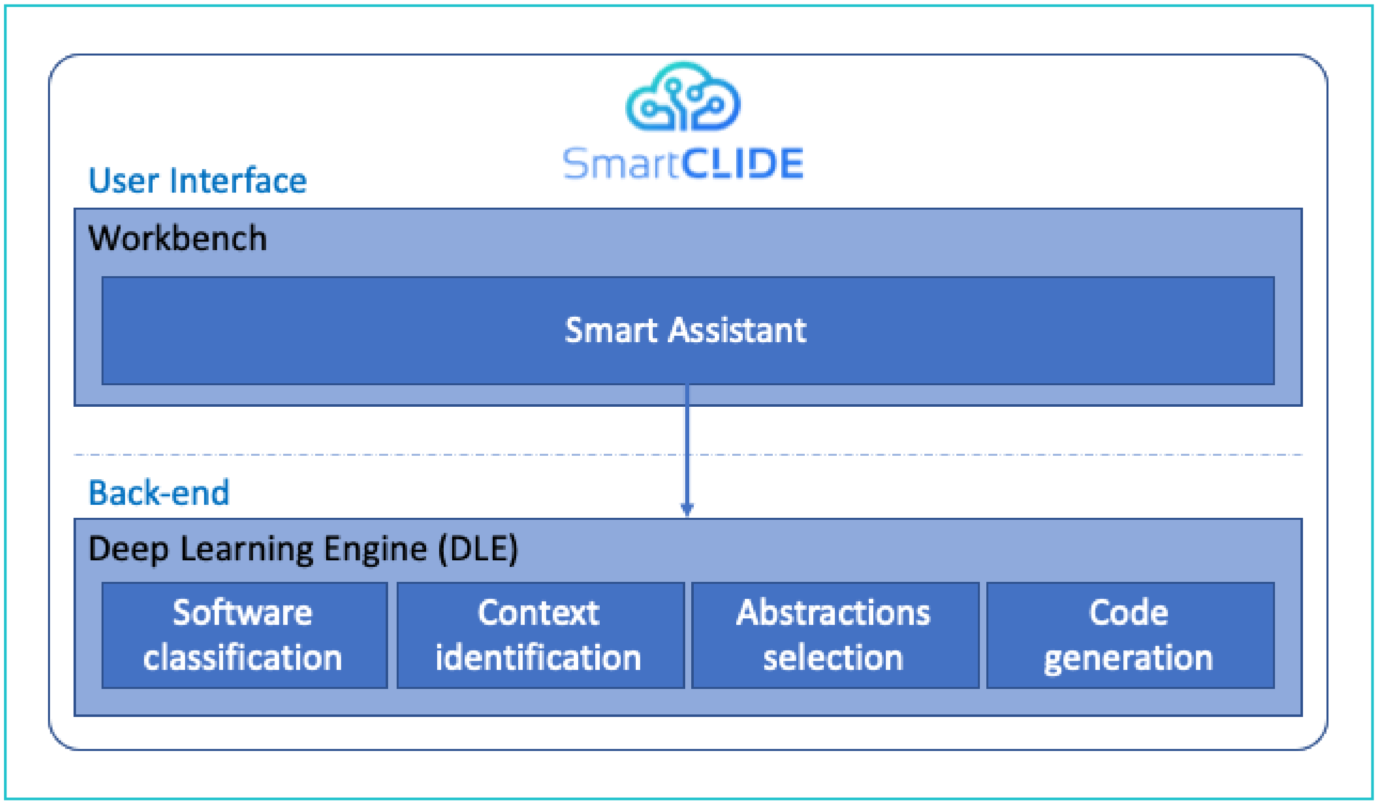


Figure 3: Deep Learning Engine and Services

Developing software can be frustrating and messy; the boilerplate code is repetitive and it may be difficult to reuse previously generated items, especially in large company environments. Nevertheless, services are a useful programming paradigm which enhances scalability and resource control, facilitating the maintenance (zero-downtime updates in continuous integration environments) processes carried out by small independent teams on the basis of their atomic functionality. Hence, SmartCLIDE proposes an assistant who will:

- Help users develop services based on BPMN (Business Process Model and Notation) schemes
- Help developers create quality code through suggestions, syntax highlighting and providing easy documentation
- Help users/developers reuse already existing services

Apart from this, a non-technical user should be capable to define a functionality and be guided through the software composition process, enabling the use of existing and previously classified services.

SmartCLIDE will research Machine and Deep Learning techniques, testing their advantages over simpler approaches, when faced with challenges in quality assessment, service composition, serviceclassification and service discovery, along with code suggestions.

In sum, a Deep Learning Engine will be designed to support ML and DL techniques at the core of SmartCLIDE. This is a big challenge in terms of project aims and the number of techniques to be tested.

SmartCLIDE: a new cloud-native IDE

By ATB Bremen

Analyzing data is much easier and faster today thanks to cloud computing and on-demand availability of computer system resources such as data storage and computing power. However, the development of cloud solutions requires tools adapted to special characteristics of the cloud and fast time-to-markets demanded by companies and organizations. Cloud distributed systems are highly complex due to the wide variety of technologies and frameworks that can be used in the whole stack, the underlying microservice architectures, or the management of the deployment pipeline. To develop services from scratch is expensive, as well as time-consuming, and one of the main challenges IT companies are facing is reflected in the difficulty to find qualified IT staff in the market to face these challenges.

Makes the development and deployment of data-intensive services for the cloud easier than before.

In this context, the **SmartCLIDE consortium** proposes the creation of a new cloud-native Integrated Development Environment (IDE) that makes the development and deployment of data-intensive services for the cloud easier than before, aiming at bridging the gap between on demand business strategies and the lack of qualified software professionals.

Which will be the main features of our IDE?

- **Lifecycle support.** A software follows a life-cycle, from the specification of features to the deployment of the solution. Each stage requires specific tools and SmartCLIDE will provide these tools to software-crafters just when they need them. For example, it will offer **Gherkin tools based** on the specification of the behavior of the services to develop and defining the acceptance criteria, enabling future automation of acceptance tests. At the development stage, SmartCLIDE will provide **data sources, data transformations, graphics visualization artifacts, or general-purpose abstractions and patterns** that can be combined to implement the above-mentioned features. And at a final stage, it can discover specific purpose containers for the deployment of the generated code.
- **Insightful source code monitoring.** SmartCLIDE IDE will implement visualization features that enable the developer to **gain**

a deeper knowledge about the source code. It will dynamically show the meaning of expressions, or the flow of code at low levels of granularity. It will allow to compare different software states that are achieved, perform changes in states that are reflected dynamically, or to create new abstractions that can be easily reused.

- **Version Control and Configuration** Management Integration. SmartCLIDE will enable integration with the most frequently used **Version Control** and **Configuration Management Systems** such as Github or **GitLab**. Following a **DevOps** and full-stack development approach, a unique repository shall be used to keep all the implementation items under version control: source code, binary files, configuration files, data, tests, virtual machines, containers, etc.

These are already cool features, but we propose on top some very nice features based on the power of **Deep Learning** that make SmartCLIDE very special:

- **Development by demonstration and text notation.** Making use of a Deep Learning Engine, and based on the current features, SmartCLIDE will automatically retrieve resources that it considers relevant for the new development. The end-user will be able to use **text notation** to enhance the description of the retrieved behavior or algorithm. Based on these new indications, the Deep Learning Engine will dynamically **propose programmatic solutions** to obtain the desired output. The environment will also enable developers to face programming tasks by manipulating abstractions straight forward, not requiring previous knowledge of the underlying language.
- **Automatic software classification.** Our Deep Learning Engine will automatically identify and classify already existing and new software abstractions that will be visualized in the IDE for re-utilization, based on the purpose or behavior defined by the end-user.
- **Continuous integration and deployment assistance.** SmartCLIDE will guide the user through each stage of the lifecycle, ensuring that the generated code has been properly tested, accurately integrated within the corresponding development branch, and automatically deployed in the selected cloud service, implementing the end-to-end responsibility of the DevOps philosophy.

SmartCLIDE will be based on **Eclipse THEIA**, the cloud version of the Eclipse IDE, offering all the necessary tools for developing in one single place. It will be a rich interface, with a vast range of features to accelerate the deployment of cloud services, improve their quality, and expand the skills of novel and experienced developers.

Service Discovery in a Nutshell

By University of Macedonia

n recent years, **Microservices** have gained in popularity, since they come with various advantages, which are very useful for contemporary software development for example, in the era of **containers, decentralization** and **cloud computing.**

Microservices can be developed and deployed on different platforms, using different programming languages and development tools.

Additionally, the microservices architecture is an approach in which an application is broken down into a number of components, in which each component is responsible for a specific role. Different components communicate with each other using network protocols (e.g., HTTP) through connectors (e.g., APIs). A cloud platform is able to provide such services to the user. The term **"cloud services"**, as described in other blog posts, refers to a wide range of services delivered on-demand to companies and/or customers over the internet. These services are designed to provide easy, affordable access to applications and resources, without the need for internal infrastructure or hardware. Ranging from checking emails to collaborative document writing, most employees use cloud services throughout their workday, whether they're aware of it or not. As a result, microservices are widespread and used by many expert or novice end-users.

In addition to that, microservices offer many functionalities that can be easily deployed or modified saving time and effort.

Nevertheless, the large variety and the number of available services make it difficult for a novice cloud developer to find and choose the right services for his/her application.

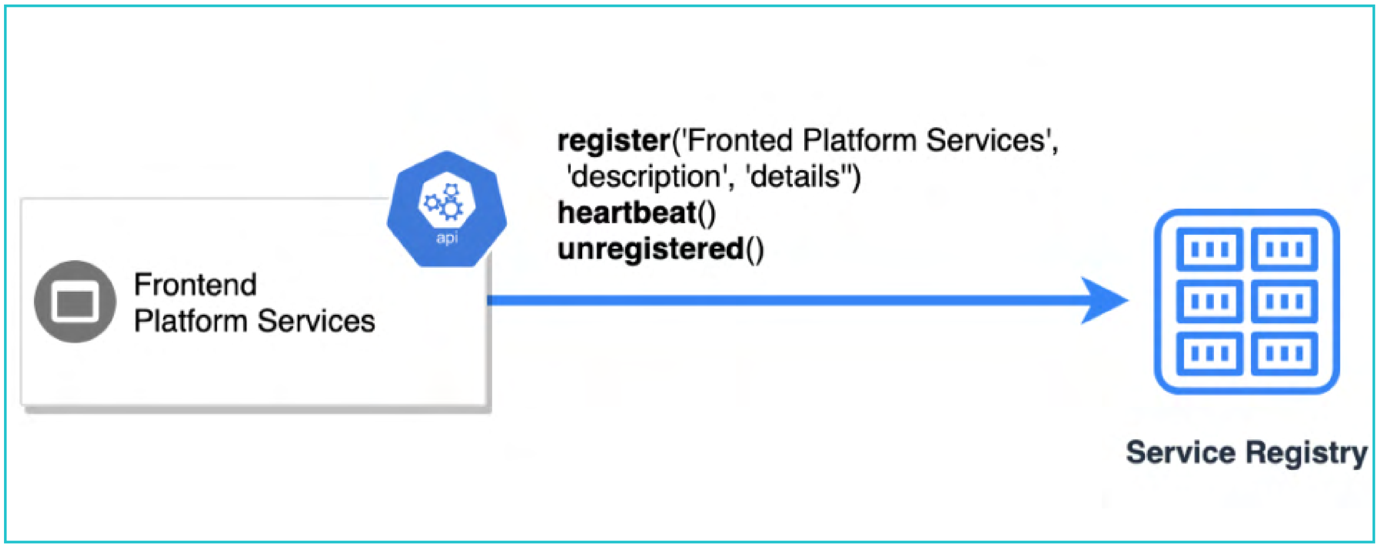
Luckily, all cloud providers as well as the research community have experimented with **Service Discovery.** Service Discovery refers to the processby which, on the one hand, an application-user learns (by search) what services are available on the network, and on the other hand, the network "learns" what services the application can provide.

How does Service Discovery Work?

There are three components to Service Discovery: the **service registry**, the **service provider** and the **service consumer.**

The Service Registry. The service registry is a key part of service discovery. A service registry needs to be highly available and up-to-date. Service instances must be registered with and deregistered from the service registry. The identification of services can rely on advanced search methods, and a search in public or private repositories, or on the internet. In a cloud platform, every service has many instances, and each instance is being used by an application or a user. In terms of adding a new service to the system, it is common to use the Self-Registration Pattern in which the Service Provider adds a new service with its information on the system. The alternative to self-registration pattern is the existence of a system component to manage the registration of service instances: namely, the third-party registration pattern. Both solutions are presented below:

The Self-Registration Pattern. In the self-registration pattern, a service instance is responsible for registering and deregistering itself with the service registry. Also, if required, a service instance sends heartbeat requests to prevent its registration from expiring.



register('Fronted Platform Services',
'description', 'details")
heartbeat()
unregistered()
api
Frontend Platform Services
Service Registry

Figure 4: The Self-Registration Pattern

The Third-Party Registration Pattern. Third-party registration allows delegation of service registration/deregistration task to Third-party registrar (service manager) component. On service instance start-up, service manager is responsible for registering the service with the Service Registry. Similarly, it de-registers on service shutdown. With this option, service resilience can be better handled as Service Manager can handle these requests more elegantly than self-registration where a service can abruptly go down with Service Registry not being aware. The following diagram shows the structure of this pattern.

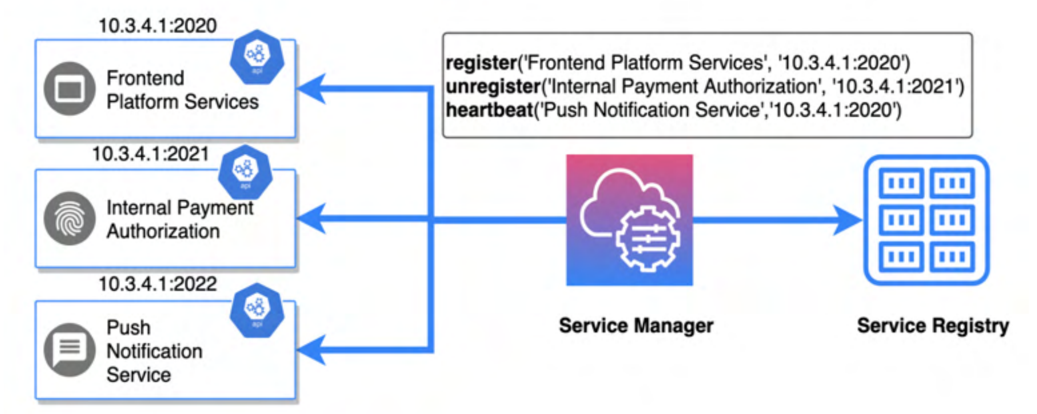
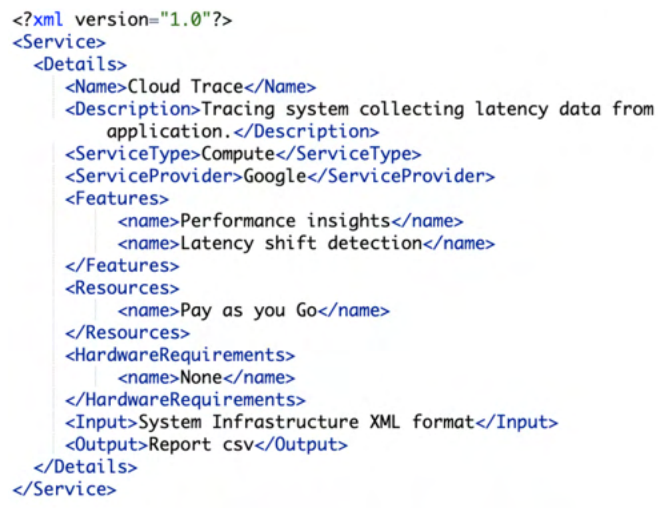

10.3.4.1:2020
Frontend Platform Services
10.3.4.1:2021
Internal Payment Authorization
10.3.4.1:2022
Push Notification Service
register('Frontend Platform Services', '10.3.4.1:2020')
unregister('Internal Payment Authorization', '10.3.4.1:2021')
heartbeat('Push Notification Service','10.3.4.1:2020')
Service Manager
Service Registry

Figure 5: The Third-Party Registration Pattern

Service Discovery. Regardless of the Service Registry decision, there is a need for a predefined protocol (i.e., a specific way to describe the service) to enable discovery. A service description captures the functional and non-functional characteristics of a service in a format that can be machine-read and processed. This description helps the Service Discovery to find services based on search criteria. A service description can be an XML file as shown below.



```
<?xml version="1.0"?>
<Service>
  <Details>
     <Name>Cloud Trace</Name>
     <Description>Tracing system collecting latency data from
         application.</Description>
     <ServiceType>Compute</ServiceType>
     <ServiceProvider>Google</ServiceProvider>
     <Features>
          <name>Performance insights</name>
          <name>Latency shift detection</name>
     </Features>
     <Resources>
          <name>Pay as you Go</name>
     </Resources>
     <HardwareRequirements>
          <name>None</name>
     </HardwareRequirements>
     <Input>System Infrastructure XML format</Input>
     <Output>Report csv</Output>
  </Details>
</Service>
```

Figure 6: A Service Description

The service discovery is a search process that aims at finding available services with specific requirements or finding instances of a service by performing queries on the service registry. There are two primary strategies for discovering services, via a service registry:

Client Side Discovery. The client contacts a service registry, receives details for available services, and contacts one of them using a load balancing algorithm. When the client requires a microservice, it finds a suitable service in the registry and connects to it directly. The assumption is that the registry tracks availability of services using a heartbeat mechanism.

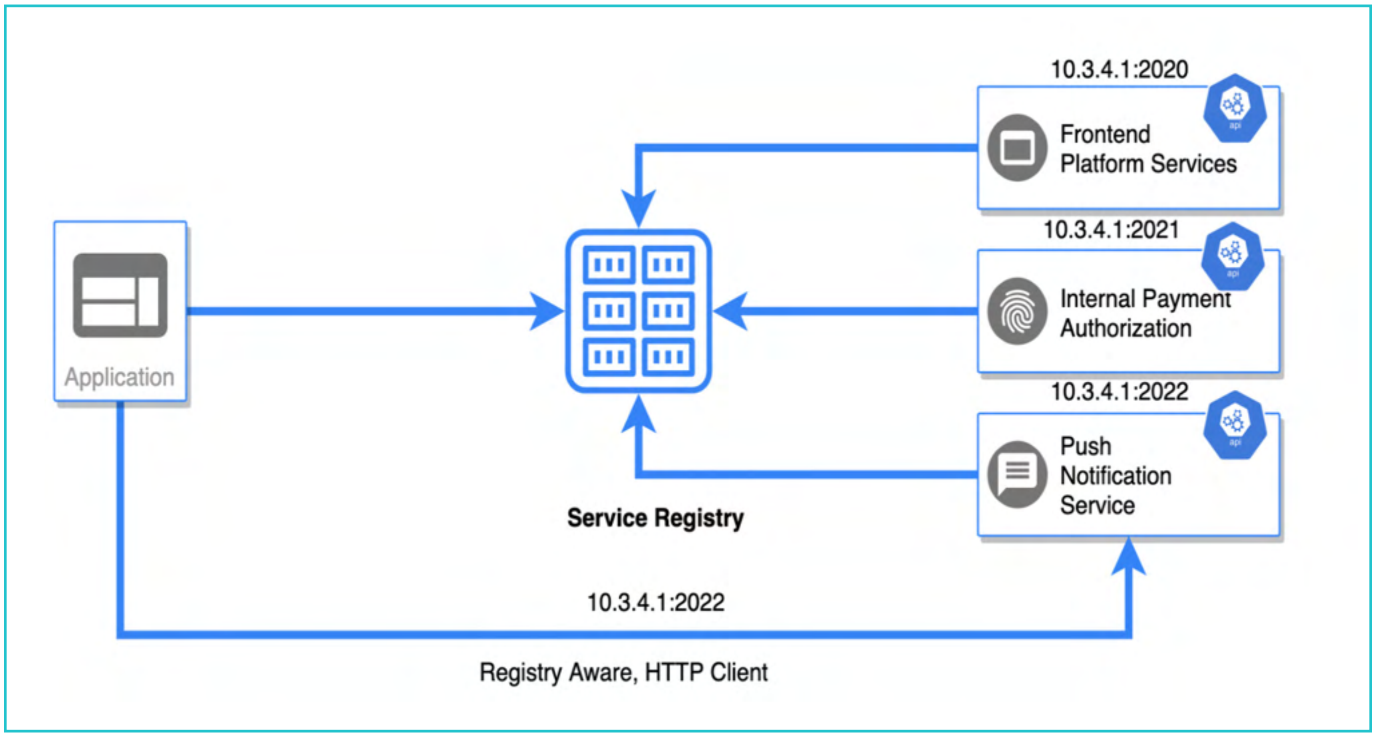

10.3.4.1:2020
Frontend Platform Services
10.3.4.1:2021
Internal Payment Authorization
10.3.4.1:2022
Push Notification Service
Application
Service Registry
10.3.4.1:2022
Registry Aware, HTTP Client

Figure 7: Client-Side Discovery

Server Side Discovery. The client contacts a load balancer, making a request that indicates which type of service it needs. The load balancer consults the service registry, selects the optimal service and routes the request to it. Load balancing is commonly used as a service discovery mechanism; it provides health checks and can automatically register/unregister services when they fail. The load balancer works in tandem with the service registry.

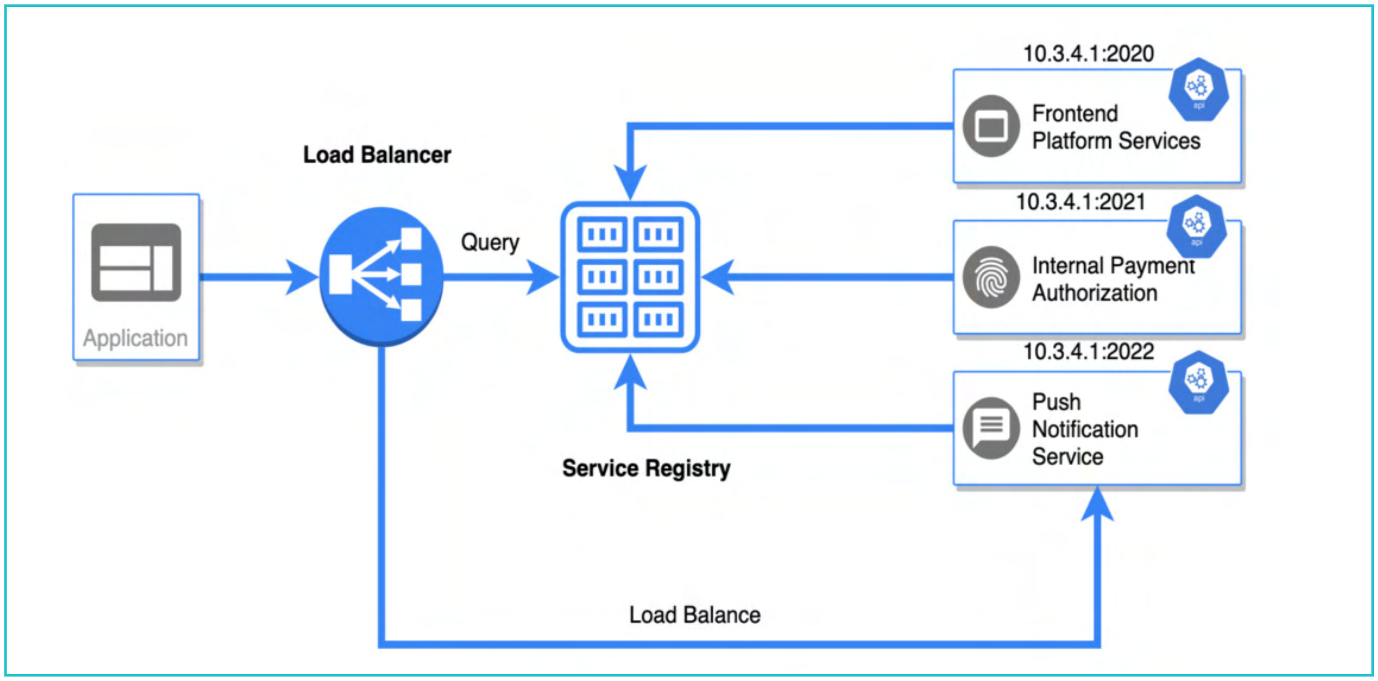


Figure 8: Server-Side Discovery

Service Consumer. Finally, these applications are being used by some entity called Service Consumer. The service consumer could be an application, a service, or any other type of software module that requires a specific functionality, through a service. This entity uses the Service Discovery to find services within the system, binding to the service over a transport protocol and then executing the service function. The service consumer executes the service by sending a request formatted according to the service description. The most common example of a service consumer is a REST Web Service call like this one:

api.mycompany.io/v1/party/findById/12

This request invokes a service of the revision **v1** that exists in the **api.mycompany.io** host. Specifically, it will try to find a **party** entity with **id=12.**

Examples of Service Discovery in Industry

1. Google API Discovery Service: The Discovery API provides a list of Google APIs and a machine-readable "Discovery Document" for each API.
2. Amazon ECS Service Discovery (AWS Cloud Map API): AWS Cloud Map is a fully managed service that can be used to create and maintain a map of the backend services and resources.
3. IBM Service Discovery API: Service Discovery is a core service within a cloud microservices architecture that can be used to accelerate the development of applications in the cloud environment.
4. Docker: Service discovery registers a service and publishes its connectivity information so that other services are aware of how to connect to the service. Some ways to achieve service discovery with docker are:
   a. Service Discovery with DNS
   b. Internal Load Balancing
   c. External Load Balancing (Swarm Mode Routing Mesh)
   d. The Swarm Layer 7 Routing (Interlock Proxy)
5. Azure API Management with microservices: As a full-lifecycle API management solution, it provides additional capabilities including a self-service developer portal for API discovery, API lifecycle management, and API analytics.

References

www.g2.com/categories/service-discovery

www.nginx.com/blog/service-discovery-in-a-microservices-architecture/

ns1.com/dns-service-discovery

www.datawire.io/guide/traffic/service-discovery-microservices/

avinetworks.com/glossary/service-discovery/

medium.com/@jamesemyn/service-discovery-in-microsrvice-cbd54afb94f3

auth0.com/blog/an-introduction-to-microservices-part-3-the-service-registry/

www.magalix.com/blog/kubernetes-patterns-the-service-discovery-pattern

cloud.google.com/service-infrastructure/docs/service-consumer-management/reference

docs.aws.amazon.com/cloud-map/latest/dg/what-is-cloud-map.html

developers.google.com/discovery

developer.ibm.com/api/view/id-129:title-IBM_Service_Discovery_API#Overview

success.docker.com/article/ucp-service-discovery-swarm

docs.microsoft.com/en-us/azure/api-management/api-management-kubernetes

Programming By Example

By AIR Institute & KAIROS DS

The aim of Programming By Example (PBE) is to develop programs through the synthesis of a series of examples. First, a sequence of actions is performed or given by the user: this is the starting point of a combination of functions which result in a programmatic output, designated for a specific task.

With this technique, users are able to create programs by interacting with the interfaces they are used to, implementing generalizations to problem-solving techniques which are independent from the data they were generated with.

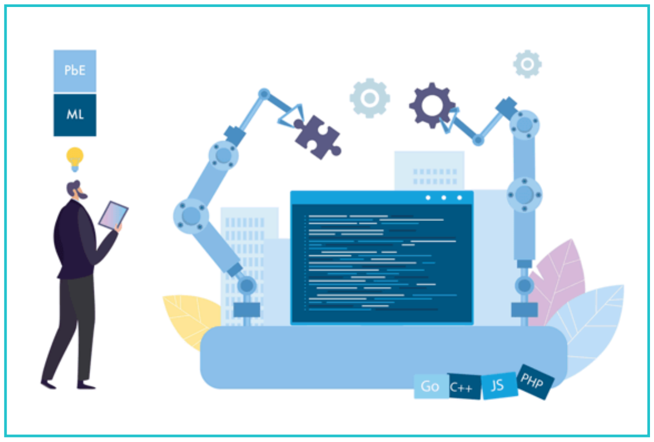
A more technical definition states that PBE is a synthesis technique by which programs are iteratively and automatically generated, from tuples of inputs and outputs called examples

In case the generated program does not operate correctly, a new tuple has to be introduced to adjust the programmatic output. Thus, users provide input/output combinations (examples) of the task they want to perform, and the computer infers a program that is capable of addressing the problem.

Although PBE is targeted at non-expert users and its purpose is to lighten the workload associated with programming, it nevertheless has added value to advanced users because it mitigates tedious and repetitive tasks, optimizing their work. Moreover, the generated code can be reviewed by a human –the output is legible and easy to understand depending on the developer-, given that a large part of the program is normally correct, only some parts are too oriented to the examples it was trained with. This characteristic means a program does not have to simply be taken or discarded as ML black-box models do. Thus, in PBE, a program can be modified if the number of examples is not sufficient.

Nonetheless, this methodology suffers from a series of limitations. For example, the generalization is not broad enough to deal with all the plausible data types, and the program is not able to cope with variations to its output.

The definition of a generic Domain-Specific Language (DSL) is key to a PBE. DSL is a grammar of production rules whose aim is to narrow down the search space; it represents the limits of a PBE system. If a program can be described in terms of a DSL, then a solution may be found. Otherwise, it is impossible, no matter how many examples are provided.



PBE is also known as inductive synthesis: a synthesis process that is based on examples. A deductive synthesis, in turn, is based on logical specifications defined by the user. PBE breaks a common programming rule, in that PBE users are not simply consumers because they can, to some degree, build their own code. This means small scripts are automatically generated for little everyday tasks. For advanced users, PBE can be a helping hand too. It's especially useful for data scientists who must normally manage big amounts of data before they can apply AI algorithms. Normally, data is obtained from diverse sources which have different degrees of structuring. While they provide users with a high level of flexibility, they make it hard to exploit, combine, and query data. Unfortunately, a major problem associated with inductive synthesis is the ambiguity which results from defining the behaviour of the program and not its exact requirements.

PBE Application and generated output

The main application fields of PBE are robotics, code refactoring, data parsing or query building, and prominently, the so-called data wrangling.

Data wrangling consists in pre-processing the data that is to be fed to other tasks. The process can be divided into three parts: extraction, transformation and formatting. Extraction consists in the generation of structured data from semi-structured sources, such as web pages or JSON files, where a program is built for every field extraction. Transformation addresses type casting and combining fields, e.g. the composition of names from several related fields. Finally, formatting means that a specific format is applied in a repetitive way or a structured output is created from the previously generated data.

Code refactoring allows users to save time on common maintenance tasks, enhancing users' time management and performance.

Regarding the code generated by PBE tools, code generation is a complex process whose results are not always satisfying: while in an ordinary sense traditional program synthesis consists in creating scripts which satisfy a series of logical conditions, in PBE, scripts are synthesised from a number of input/output states. This model is successful because it allows users to define the desired behaviour, and then, tune-up the result manually. On the other hand, it is difficult to generate code that is consistent with all the examples provided by the user.

The absence of a sufficient number of examples is a common problem during program composition. It is tackled by applying techniques such as Machine Learning (ML), which make it possible to rank intermediate functions, or to extract feedback on the generated programs.

In addition, more complex programs can be created. These programs use sequential predefined functions to perform specific tasks. This is done in AI approaches whose aim is to enhance the results of a PBE process, obtaining programs which solve high-level functions from some simpler, atomic ones.

ML vs PBE

The relationship between ML and PBE is complementary, although, in some cases, they may be used separately to deal with similar problems. While both use example data to produce specific-purpose code, the main difference lies in PBE's suitability for small repetitive tasks:

- PBE programs may be edited and adapted after they have been generated, to ensure they are fit for their final purpose. This can be done to optimize and adjust their functioning. On the contrary, ML models may only be applied to data.
- PBE requires a lesser amount of data (examples) to infer a generalization, and in this manner generate a proper output.
- ML can be used to enhance the PBE process of program generation, making the search for an ideal function faster. Also, PBE makes it easier to tackle tasks that must be performed prior to the application of AI algorithms. Note that ML has to be used to deal with complex data tasks.
- Some researchers have made efforts to apply Neural Networks to PBE code generation, by means of the aforementioned process of using sequences of atomic particular-purpose functions to achieve a complex result.

Programming By Example in SmartCLIDE

The inclusion of the PBE paradigm in **SmartCLIDE** could help create generalizations which would spare the user certain development tasks during the creation of services, and provide help and support with the tasks associated with the **Deep Learning Engine** (DLE), such as data pre-processing.

The combination of PBE with some other concepts like **Context** will be fundamental for the simplification of the development process for non-technical users. This will also help make service generation easier for developers. PBE represents too an interesting feature to test its matching possibilities with AI usage.

To sum up, PBE has to be tested as a helper in coding and data processing. The SmartCLIDE interface could offer the benefits of PBE, reducing the workload associated with complex syntaxes and repetitive tasks.

OUR SCENARIOS OF USE

This second section presents scenarios where SmartCLIDE will be validated and evaluated under real conditions. There are 4 such scenarios:

- **Wellness Telecom** proposes a real-time communication project that involves the deployment of multiple virtual machines, providing a compelling use-case for SmartCLIDE at the creation of run-time abstractions like real-time constraints of the communication process and the validation of the deployment in software-defined infrastructures.
- **UNPARALLEL** proposes two different scenarios for SmartCLIDE piloting its evaluation in the evolutive development and interfacing of an IoT web catalog with SmartCLIDE, enabling the end-users of the portal (mostly IoT developers or integrators) to develop IoT solutions with SmartCLIDE.
- **CONTACT Software** proposes to evaluate SmartCLIDE as part of its ELEMENTs integration platform, enabling potential customers to build their own IoT-related services.
- **Netcompany-Intrasoft** will make use of SmartCLIDE at all the stages of the lifecycle within an existing software project.

If you would like to know more about our project, we invite you to visit the SmartCLIDE.eu website and subscribe to our newsletter to receive regular updates on our progress.

About LoRaWAN communication services

By Wellness Telecom

LoRa is a radio modulation technique that is essentially a way of manipulating radio waves to encode information using a chirped (chirp spread spectrum technology), multi-symbol format. LoRa as a term can also refer to the systems that support this modulation technique or the communication network that IoT applications use.

The main advantages of LoRa are its long-range capability and its affordability. A typical use case for LoRa is in smart cities, where low-powered and inexpensive internet of things devices (typically sensors or monitors) spread across a large area send small packets of data sporadically to a central administrator.

A low-power wide-area network (LPWAN) is a type of wireless telecommunication network that allows connected devices to have long-range communications capabilities at a low bit rate. LPWANs are typically used in asset monitoring and management in smart cities and Industrial Internet of Things deployments. This is in contrast to wireless wide-area networks (typically used by large corporate organizations) that carry more data and use more power. Examples of LPWAN technology are Lora/LoraWAN, Sigfox, MIoTy, Wi-SUN, LTE-M, and NB-IOT.

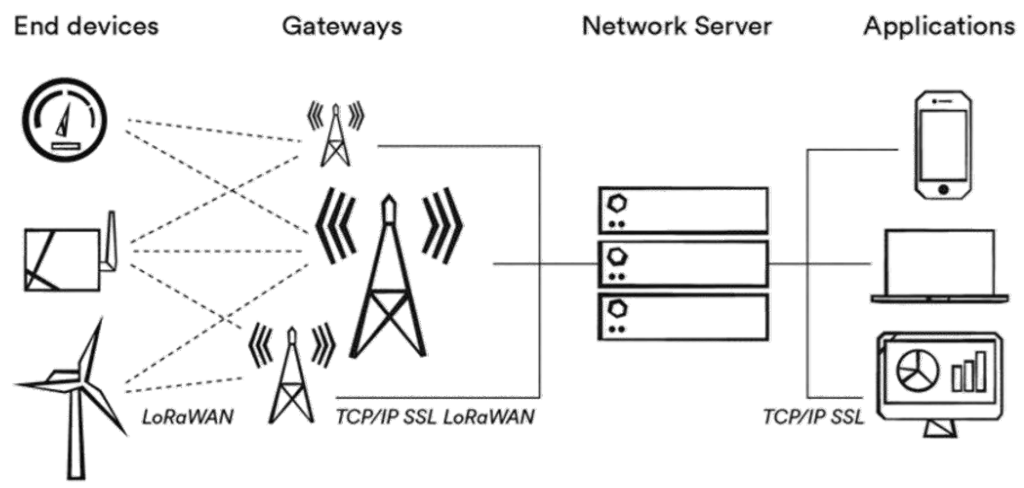


Figure 9: A low-power wide-area network (LPWAN)

The Concept

Wellness TechGroup proposes a microservices-based LoRaWAN communications platform to face all these difficulties. When compared to old monolithic applications, microservices can provide: (a) replication of microservices (i.e., to improve availability, adaptability and scalability), and (b) an inherent security layer thanks to container and virtual machine (VM) native isolations.

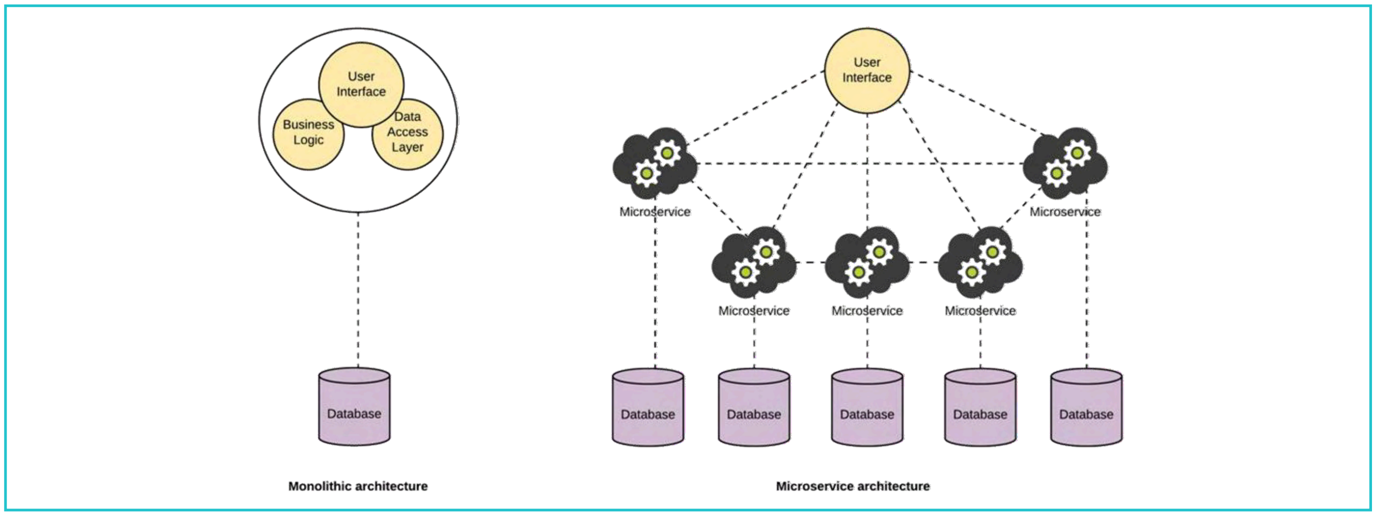


Figure 10: Monolithic vs Microservice architecture

Despite its usefulness, the microservice architecture presents a series of difficulties for the developer given the large granularity of its elements. For example, in the chosen real-time communication use case, the final system must include, at least, the following functionalities (as separated microservices):

- ChirpStack Network Server is responsible for handling (and de-duplication) of uplink data received by the gateway(s) and the scheduling of downlink data transmissions.
- ChirpStack Application Server is responsible for the node "inventory" part of a LoRaWAN infrastructure, handling of received application payloads and the downlink application payload queue
- ChirpStack Gateway Bridge is a service which converts LoRa® Packet Forwarder protocols into a ChirpStack common data-format (JSON and Protobuf).
- Mixer and/or transcoder: supplies transcoding and mixing video tools.
- Database (e.g., MySQL, Redis…): supplies storage capabilities (e.g., registered users, calls in progress)

Docker container Kubernetes-based deployment

Containerization involves the packaging of code and its dependencies together. To better understand containerization with Docker and Kubernetes, this guide provides an example of developing a simple application, containerizing, and deploying it to a Kubernetes cluster.

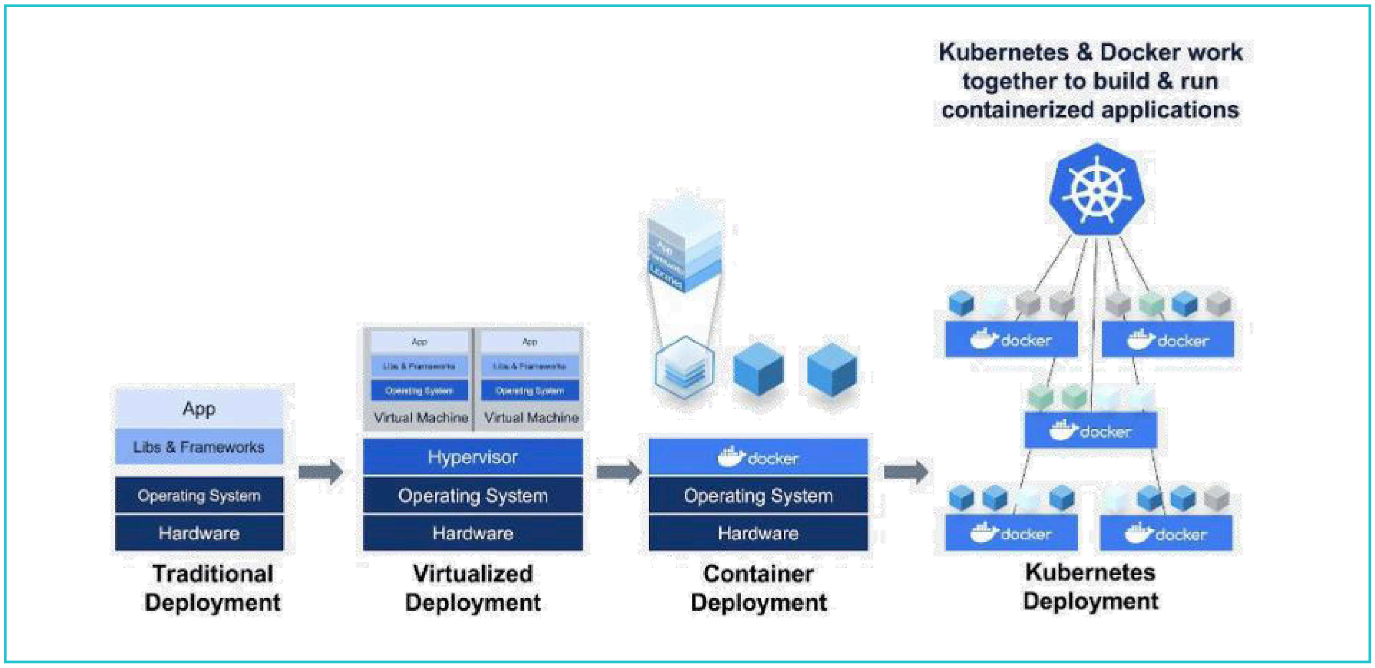


Figure 11: Deploying an application to a Kubernetes cluster

Consequently, the whole LoraWan communication platform system can be illustrated as:

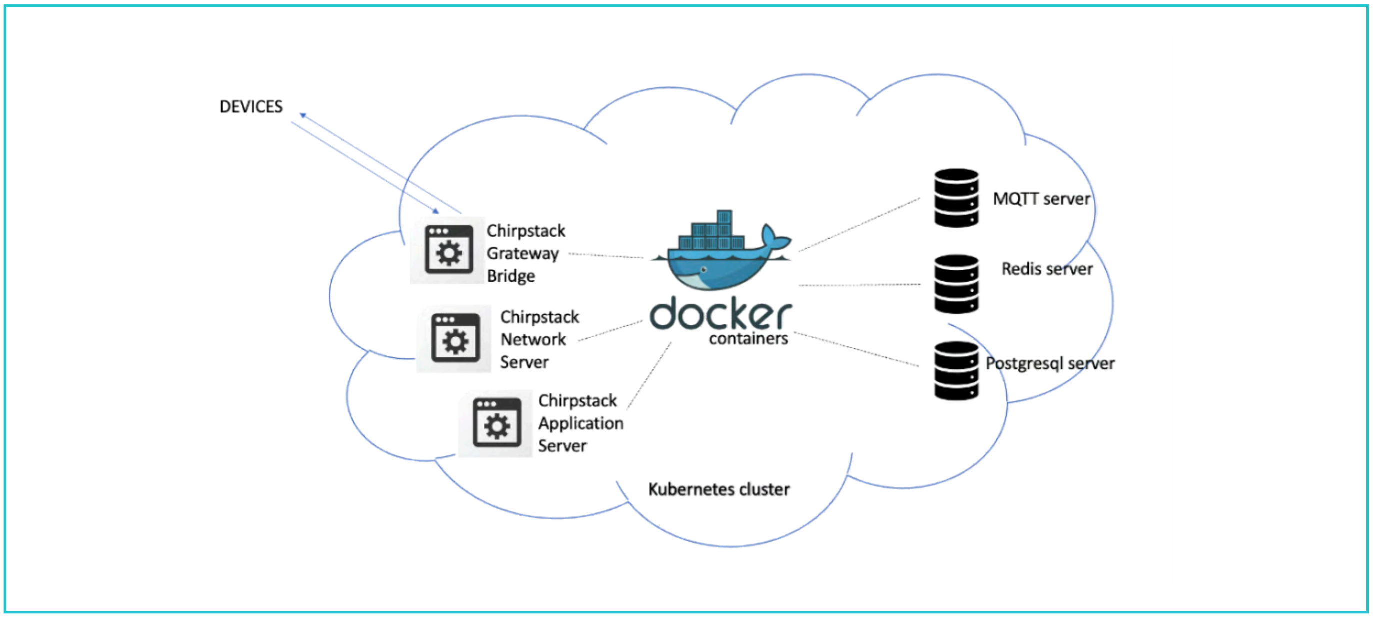


Figure 12: : Real-time communication platform system

SmartCLIDE benefits

Developing, deploying, and monitoring complex systems such as the Chirpstack LoRaWAN platform we previously described using containers is a tough task, but thanks to SmartCLIDE, developers will be able not only to deploy it, but also to get:

- A better comprehension of real-time deployment costs and time

- A deeper insight into deployment costs, by providing monitoring tools that will allow the developer to track the cost of a deployment for major cloud providers.
- Easier management of deployed services.
- Smoother service delivery through reduced number of errors and reduction in time to resolution
- Improved understanding of lifecycle costs
- Overall increase in agility and efficiency of initial deployments

In conclusion, SmartCLIDE will aid the developer throughout every phase, such as development, testing, and deployment. Then, once deployed, the IDE (Integrated Development Environment) will supply visual monitoring tools to manage and extend running capabilities, also providing a detailed analysis about deployment and deployment costs.

In conclusion, SmartCLIDE will aid the developer throughout every phase, such as development, testing, and deployment.

Enhance IoT-Catalogue with an integrated Cloud IDE

By UNPARALLEL

IoT-Catalogue (iot-catalogue.com) is the one-stop-source for Internet-of-Things innovations and technologies to help users (developers/integrators/advisors/end-users) take advantage of IoT for the benefit of society, businesses, and individuals. IoT-Catalogue provides a large range of information about **products,** including both hardware devices and software components. It also identifies **solutions** designed to target specific **ICT problems** and describes **use cases** where solutions were applied.

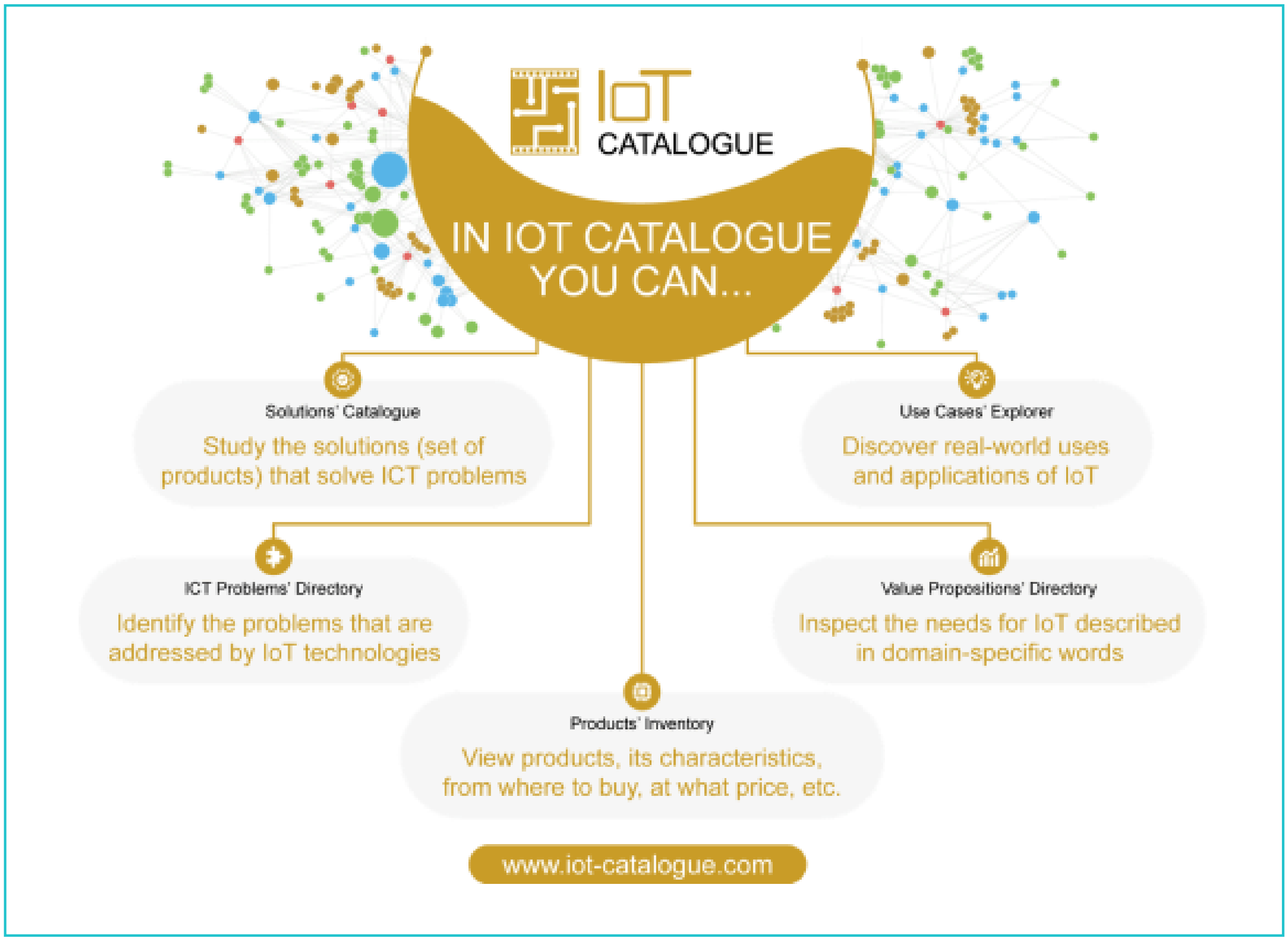

IoT CATALOGUE
IN IOT CATALOGUE YOU CAN...
Solutions' Catalogue
Study the solutions (set of products) that solve ICT problems
Use Cases' Explorer
Discover real-world uses and applications of IoT
ICT Problems' Directory
Identify the problems that are addressed by IoT technologies
Value Propositions' Directory
Inspect the needs for IoT described in domain-specific words
Products' Inventory
View products, its characteristics, from where to buy, at what price, etc.
www.iot-catalogue.com

Figure 13: IoT Catalogue

IoT-Catalogue holds and presents a wide range of information that can help IoT developers design their IoT solutions. Options range from low-level hardware (like boards and sensors) to platforms and services to store and process data. However, the current IoT-Catalogue lacks mechanisms to make use of the available information to actively support the developers in the creation of IoT applications.

In this sense, an integrated IDE that provides abstractions for the tools and services listed in IoT-Catalogue would enable the development of users' own IoT solutions.

SmartCLIDE benefits

The integration of SmartCLIDE technologies in IoT-Catalogue will evolve IoT-Catalogue to a closed-loop IoT development platform that:

- Supports IoT developers on the design and implementation of their IoT solutions;
- Helps developers select the hardware that better suits the intended purpose;
- Supports the development of the IoT solution software that will use the hardware selected;
- Allows the definition of the complete behavior of the solution, from the data acquisition on devices to the data storage and processing on platforms in the Cloud.

Provide a Quick Demonstration for a Customer

By CONTACT Software

The SmartCLIDE project has based its use cases on a set of 29 generic use cases, which 5 Pilot Cases demonstrate and make use of. The Pilot case by **CONTACT Software,** described here, involves the scenario where a service provider is working with customers throughout the lifecycle of a custom solution.

30 YEARS PIONEERS
Focus
Innovations for digital product processes for 30 years
Mission
Strengthening employee collaboration and making companies more agile
Industries
Customers with challenging processes
Mechanical and plant engineering
Automotive
High-tech & Medical
Consumer
~400 Employees
More than 25% R&D investments
Global Ecosystem
Community with 2000 customer locations in 40 countries
60 partners

CONTACT Software is a leading provider of solutions for **CAD data management, product data management** (PDM), and **product lifecycle management** (PLM). Founded in 1990, the objectives of our founder, Karl Heinz Zachries, remain our objectives of today: making complex product data more accessible and connecting employees across technical and organizational boundaries.

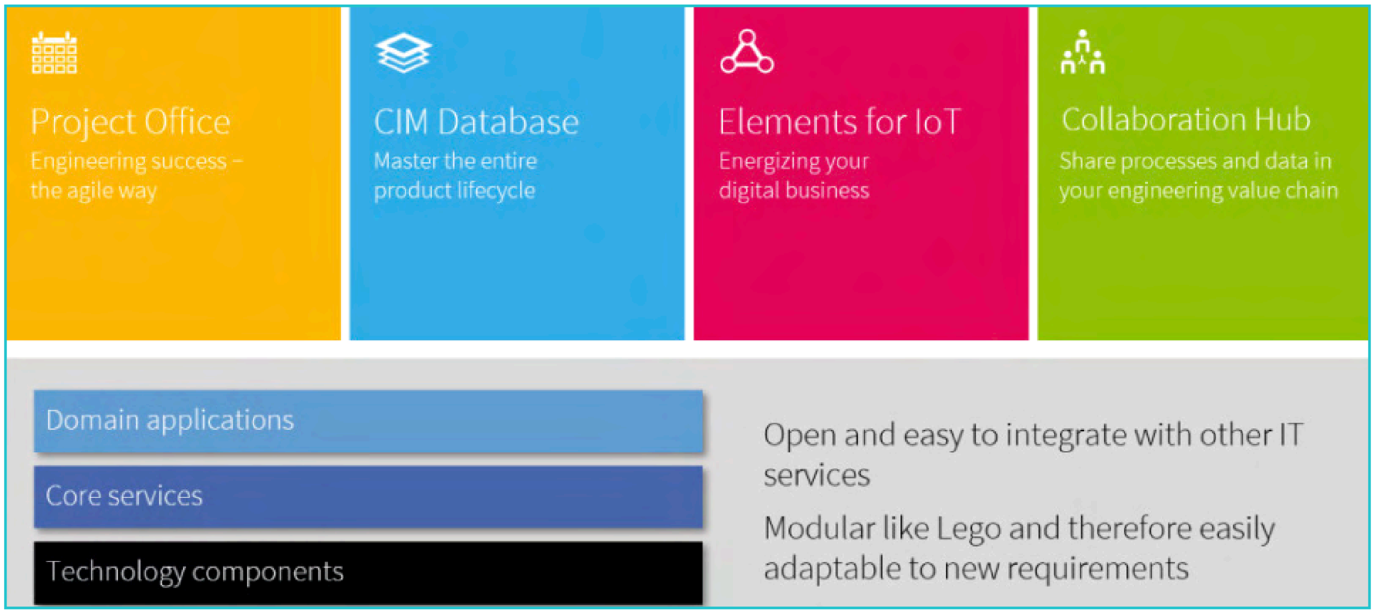
“Making complex product data more accessible and connecting employees across technical and organizational boundaries.”

CONTACT is, at its core, a customer-focused company. We act in accordance with market requirements and, above all, the requirements of our customers. We maintain long-term partnerships with our customers according to our motto “Continuity and Perspective”. Our team has expertise in industry and industrial processes, PDM/ PLM technology, and project implementation.

CONTACT employs about 400 employees with headquarters and product development in **Bremen, Germany**. Our substantial R&D commitment ensures that we know what topics will be relevant tomorrow and develop – often together with our customers – appropriate solutions early on, for example, at the moment for the wide range of topics relating to Industry 4.0 / Internet of Things. We provide our customers with comprehensive support in concept development and implementation, replacing old systems and supplementing existing installations in the face of additional requirements.

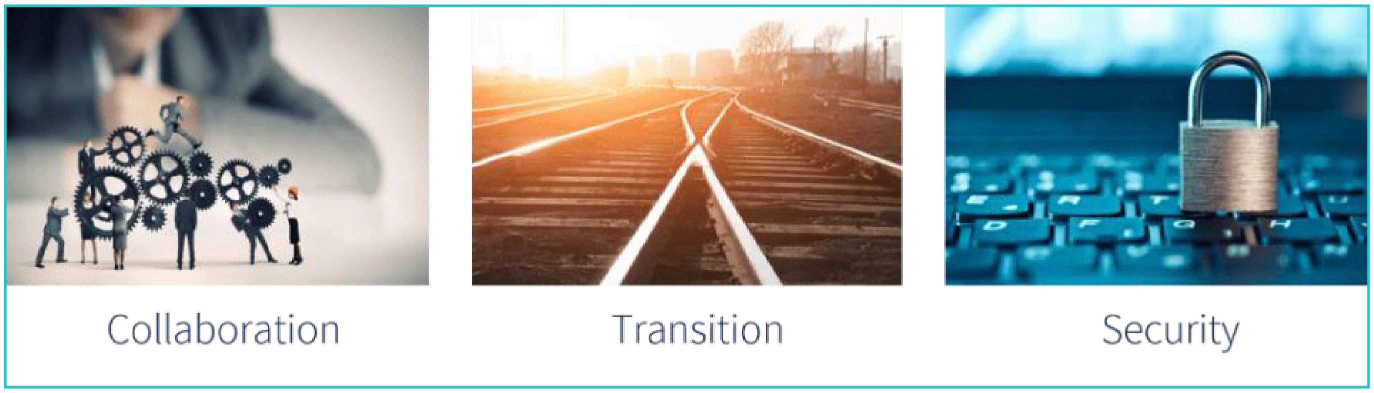
As an overall provider, we feel responsible for the complete solution. Our customers benefit from coordinated, one-stop consulting, implementation, and technology. Our customers are manufacturing companies and development organizations that often operate worldwide and rank among the leaders in their market segments.

Design principle: modular instead of monolithic


Project Office
Engineering success – the agile way
CIM Database
Master the entire product lifecycle
Elements for IoT
Energizing your digital business
Collaboration Hub
Share processes and data in your engineering value chain
Domain applications
Core services
Technology components
Open and easy to integrate with other IT services
Modular like Lego and therefore easily adaptable to new requirements

The foundation for our design principle, **“modular instead of monolithic”**, is provided by the CONTACT Elements platform. Similar to Lego, CONTACT Elements’ modular design principle allows comprehensive applications to be created that are more than the sum of their parts: each outstanding in its own right, together something unique. The same is true of its consistent ease of use and thus outstanding user experience. Our open standard solutions support the product lifecycle from the initial idea through to its deployment at customer sites. Our open standard solutions are CIM Database PLM, Project Office, Collaboration Hub, and Elements for IoT. We combine data management for virtual products with the functions required for collaboration and for process and project management in product development. We offer an extensive library for industrial IoT services, with modules for the digital twinning of real products.

We live in a world of innovative companies that are working on the products of the future each and every day and are putting the digital transformation to smarter products into motion. Our priority is to consider in-depth, listen carefully, and offer advice on an equal footing with our partners.



CONTACT Software is currently in a transition phase in its architecture, from a client/server installation on-premise to a cloud-based installation, with customers using only web clients, thus having all of the functionality in a browser. Of course, one of the main focuses is to make the transition as smooth as possible.

However, the tools for setting up and orchestrating a diverse and complex cloud architecture are not as developed as some software engineers hoped them to be.

Especially the support of state-of-the-art methodologies in IDEs in use at CONTACT lacks depth. Thus, a great need in the current phase is to be able to simplify not only the deployment and management of cloud (micro-)services but in fact their development as well.

As a result of the openness of our products and our goal to collaborate with customers in the development and improvement of our software, we need tools that support the user and let him work as efficiently as possible. For example, when customizing a solution to their needs, we may need tools to support the creation of additional services and deployments. In addition, all parties in the collaboration are intensely aware of the need to create and use secure software both cloud-based and on-premise. Therefore, tools that support the secure development and creation of (cloud) services are of utmost importance.

Working with SmartCLIDE

For us, SmartCLIDE will be a catalyst when working with cloud technologies.

It will be a great help in sales scenarios, when the demand for a quickly deployable, but stable demonstration of a custom-tailored solution is high (depending on the requirements of the potential customer) and also in later steps of the collaboration with a customer. With SmartCLIDE we want to collaboratively work on our solution with each customer and develop it further in order to make it the best possible fit for the individual end user’s needs. In order for this to work properly, we will rely on the easy-to-use functionalities of SmartCLIDE and the low-level programming examples.

SmartCLIDE will allow the user to monitor the quality of written code and deployed services. We aim to use this functionality not only to increase our code quality and the corresponding test coverage, but we will also use it as a tool to help monitor performance of deployed services in a live production environment, both on customer premises and inhouse. Specifically, SmartCLIDE will help us to detect and avoid possible bottlenecks and breakages.

Optimizing Resources

By Netcompany-Intrasoft

Netcompany-Intrasoft is a multi-national software and IT services company, part of Netcompany, that employs more than 2800 professionals. Its Product Development Department has been developing highly customized, complex software products using the latest technology for over 20 years. These products are used in the banking, law, customs, social security, and taxation sectors, among others. One of these software products is PERSEUS©, a highly Configurable & Scalable Software Product, built upon an Open Architecture Technology, that fully automates the business processes within a Social Security and Pensions Administration Organization.

Netcompany-Intrasoft leads one of the envisioned SmartCLIDE pilot scenarios that aims to involve its PERSEUS Software Product Development teams, currently comprising of three agile teams, who will use the enhanced SmartCLIDE platform to develop new PERSEUS functionalities, with the utmost goal to evaluate the optimization levels that can be reached with respect to use of resources and development time, enhancing thus the entire software development lifecycle processes and improving team collaboration…

The need emerges primarily by, on the one hand, the observation that due to a number of commonalities in features/functions/processes in the PERSEUS sub-projects managed by different agile teams, code could be re-used across these product development teams. In addition, these teams are many times tasked to perform common parallel tasks and even repeat trivial tasks, thus optimizations in these processes could lead to both time/resources optimization as well as increased efficiency.

As PERSEUS is a multi-module system and in addition Netcompany – Intrasoft has a wide range of products available or under development, there is a rich library of potentially reusable, software components and services developed by diverse agile development teams. Thus, software re-utilization and optimized team performance and collaboration are key targets for the company. Thus, in the context of the PERSEUS pilot in SmartCLIDE, the extent of **re-utilization of software**, leading to optimized performance and collaboration of the different agile teams, will be assessed both in the case under which the SmartCLIDE platform is used and when it is not, in order to evaluate the gains. This assessment will be accomplished by gathering feedback from the involved developer teams during the pilot and at the same time measuring the number of code blocks that are re-used in each case.

Another target to be achieved concerns the **reduction in time for resolving errors.** In the PERSEUS project, the JIRA issue tracker is used for handling Features and Bugs. The Developer work is divided into two-weeks sprints and every ticket (issue) has story points estimated by the development team, before the beginning of a sprint, depending on the complexity and the time needed to resolve each ticket. At the end of every sprint, a report chart is created which shows the work that should be completed and the work that was actually completed. Using this methodology, in which each issue is tracked, the time in resolving issues is measured. This will be performed both in the case in which the SmartCLIDE platform is used by the developer teams and when it is not, to assess the gains brought by SmartCLIDE.

In addition, **the reduction in time to deploy a significant feature requested by an end user** is important. In PERSEUS, Jenkins is used to deploy its modules in slots. Before each slot, the desired wars are gathered to deploy in a list and a Jenkins job is initiated. Depending on how many wars are in this list, the time varies. Thus, the time will be measured to deploy a significant new feature using feedback from the agile team both in the case in which the SmartCLIDE platform is used and when it is not.

Furthermore, another target of the pilot is the **reduction in lifecycle costs**. This will be measured on feedback collected from the involved agile teams regarding:

- The time needed to specify what is needed to be implemented and how;
- The time spent on development by the development teams and testing by the QA testers, using the Jira issue tracker;
- The time spent on deployment (e.g. with Jenkins), both in the cases in which the SmartCLIDE platform is used and when it is not.

Finally, another aim is to detect (and thus resolve) as many security vulnerabilities in the developed software as possible to increase the security level of the produced software of PERSEUS (and other products). Thus, another metric to measure is the level of **increase in the number of detected security vulnerabilities.** This will be measured using SonarQube and feedback gathered from the involved agile teams before and after using the SmartCLIDE platform-related functions.

DEEP DIVE

This section dives into the heart of the matter: the benefits of the SmartCLIDE project. To this end, it includes 3 articles:

- The first article presents the team's approach to listing the challenges the project wants to solve and the associated proposed solutions;
- Based on the market requirements, the second article explains the added value of an architecture based on microservices. This article is divided into 2 parts:
- Part 1: The road to microservices
- Part 2: Quality and security in a microservices world
- The third article looks at a key feature of SmartCLIDE: service creation and how SmartCLIDE will support it.
- The fourth one explains the SmartCLIDE deep-learning engine
- The last one presents the SmartCLIDE User Interface

SmartCLIDE Innovative Approaches

By University of Macedonia

This article describes novel approaches for service discovery, and classification. This approach is detailed in the deliverable D2.1 “SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment“[1]. It presents the approach adopted by the project team to define and organize the project tasks.

The approach used by the SmartCLIDE team is the engineering cycles, as described in the design science framework [2]. According to Wieringa, every engineering problem can be treated as a 4-step process:

7. **identifying** the need and specifying the problem;
8. **design** the proposed solution;
9. **evaluate** the proposed solution; and
10. **apply** the solution.

The deliverable D2.1 aims at the first 3 steps of the approach in the sense that the application of the solution falls into “Assessment of SmartCLIDE at pilot users”. In this article, we focus on the 1st step identification of problems; whereas the 2nd (proposal) and the 3rd (evaluation) steps will be presented in the final deliverable D2.1.

The identification of the targeted problems stemmed from the project proposal and the associated requirements. Each problem has been decomposed into simpler tasks that need to be fulfilled to solve the problems. The tasks are decomposed into two main categories:

- technological tasks aim to solve problems by reusing or adapting existing solutions. The advancement that technological tasks is the introduced level of automation, as well as the composition of existing solutions into processes.
- research tasks aim to solve problems that cannot be treated with existing solutions; thus, urge for novel algorithms, prototypes, and tools.

The outcomes of these tasks are organized into seven types:

1. reports correspond to documents describing existing tools, repos, approaches, etc.;
2. datasets correspond to collections of data points that will be stored in a repo;
3. schemas correspond to documents describing the format of data stored in repos, or exchanged between components;
4. tool corresponds to existing implementations that solve a practical problem;
5. approach corresponds to novel research approaches (method, algorithm, etc.) for problem solving. They are expected to be

linked to a publication;

6. draft prototype corresponds to the proof-of concept implementation of novel approaches. This version of the implementation is used only for research purposes. We expect a low level of automation and no integration at this stage;

7. final prototype corresponds to the functionally final version of the research prototype. This version will be fully automated, no integration.

Figure 14 maps the problems identified and the associated requirements to a list of tasks, so as to guide the organization of the D2.1 deliverable and the research activities.

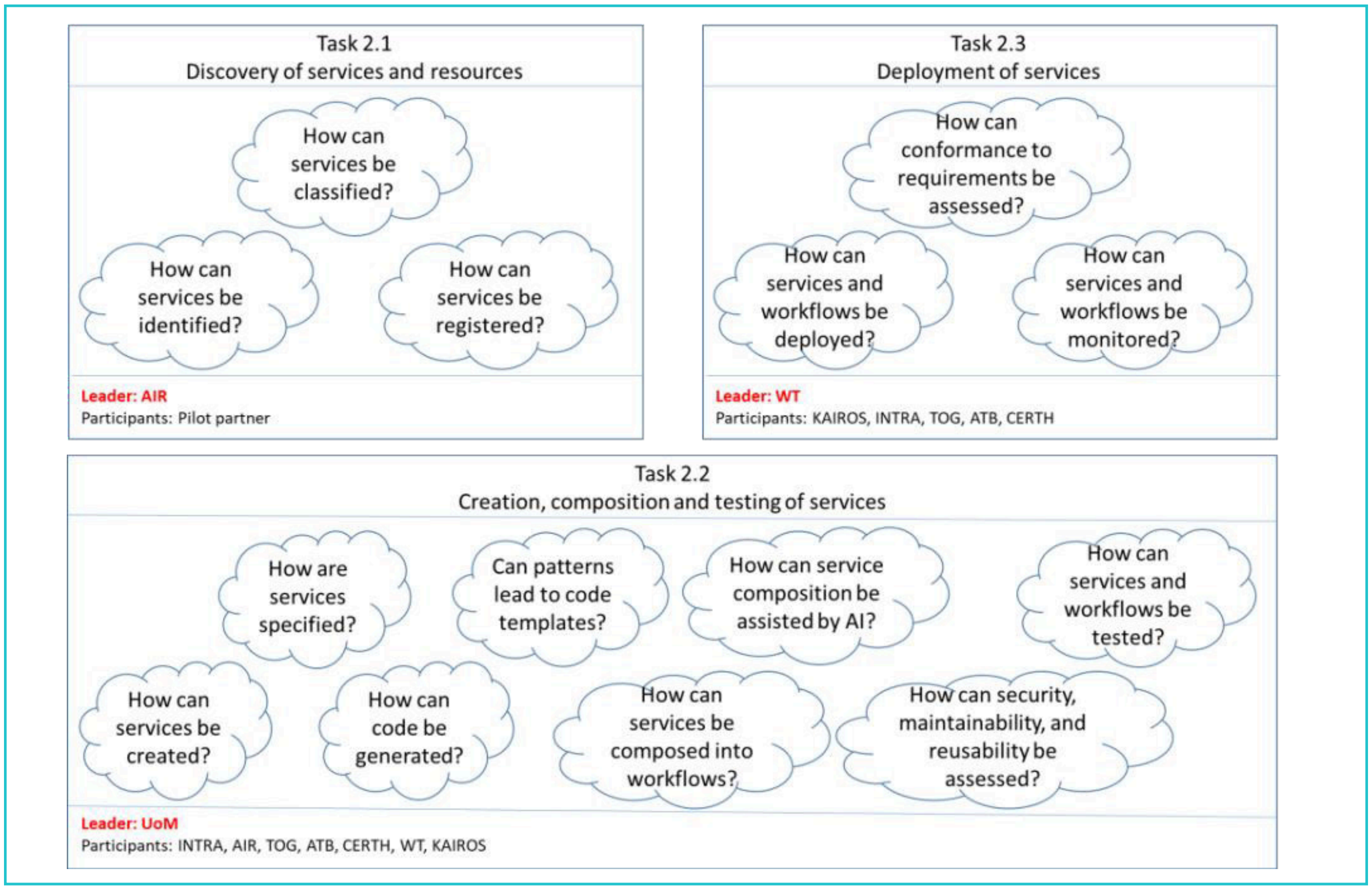

Task 2.1
Discovery of services and resources
How can services be classified?
How can services be identified?
How can services be registered?
Leader: AIR
Participants: Pilot partner
Task 2.3
Deployment of services
How can conformance to requirements be assessed?
How can services and workflows be deployed?
How can services and workflows be monitored?
Leader: WT
Participants: KAIROS, INTRA, TOG, ATB, CERTH
Task 2.2
Creation, composition and testing of services
How are services specified?
Can patterns lead to code templates?
How can service composition be assisted by AI?
How can services and workflows be tested?
How can services be created?
How can code be generated?
How can services be composed into workflows?
How can security, maintainability, and reusability be assessed?
Leader: UoM
Participants: INTRA, AIR, TOG, ATB, CERTH, WT, KAIROS

Figure 14: SmartCLIDE research problems identification

Upon problem identification, the project team proceeded to the assignment of responsibilities for answering each problem (see Table 1).

Table 1: Micro-planning per task and relevant problems

Problem	Solution
How can services be classified?	Research approach on getting services from classic registries Research approach on getting services from web pages Research approach on getting services from code repositories

How can services be classified?	Research approach on available datasets Research approach on classification model implementation
How can services be registered?	Research approach on registry service query Research approach for interfacing service registry
How can services be created?	Technological Approach on the Integration of Version Control in SmartCLIDE Technological Approach for service creation in SmartCLIDE
How can services be specified?	Technological Approach on Functional Service Specification Technological Approach on the Specification of Service Runtime Monitoring & Verification
How can code be generated?	Research Approach on Source Code Generation Research Approach on Autocomplete Suggestions
Can patterns lead to code templates?	Research Approach on Design Patterns Default Implementations Research Approach on Architectural Patterns Default Implementations Research Approach on Security Patterns Implementations
How can services be composed into workflows?	Technological Approach on Service Composition Representation Using BPML Technological Approach on Service Composition (either Discovered or Created) Technological Approach for Mapping Services to Containers
How can service composition be assisted by AI?	Research Approach on Autocomplete Suggestions on Service Composition Research Approach on Workflow Context Identification
How can services and workflows be tested?	Technological Approach for Coverage of Created Services Technological Approach for Unit and Acceptance Testing Research Approach for Automated Test Case Generation (Virtual User for Testing)

How can security, maintainability, and reusability be assessed?	Research Approach for Security Assessment at Service Level Research Approach for Security Assessment at Workflow Level Research Approach for Maintainability Assessment at Service Level Research Approach for Maintainability Assessment at Workflow Level Research Approach for Reusability Assessment at Service Level Research Approach for Reusability Assessment at Workflow Level
How can services and workflows be deployed	Technological Approach on Deployment and Orchestration Tools Technological Approach on Management Tools Research Approach on Continuous Delivery
How can conformance to requirements be assessed?	Research Approach on Cost Analysis Research Approach on Scalability Assessment
How can services and workflows be monitored?	Technological Approach on Runtime Monitoring and Verification of Services Research Approach on Defining Sensors/Metrics for Security Monitoring Technological Approach on System Monitoring (Performance and QoS)

The deliverable D2.1 details each problem and how it is planned to be solved.

References

[1] The link to the deliverable D2.1 will be available as soon it will be accepted by the project reviewers

[2] R. J. Wieringa, “Design science methodology for information systems and software engineering”, Springer, 2014.

SmartCLIDE Market Requirements

By University of Macedonia

The road towards microservices

The SmartCLIDE consortium pursues the design and development of a Cloud IDE that offers full support to the services creation life cycle: from specification of user stories to deployment in the cloud. Having performed a retrospective look to software development approaches, the consortium aims to employ a low-code software development paradigm for creating reusable and easily deployable microservices that can be indexed and composed in more complex business processes, in an online development platform that will be comprehensible for business stakeholders with limited technical skills. Artificial Intelligence methods will further assist the proposed Development Environment by providing smart auto-complete and service discovery features.

Before digging into the details of the solution, let's explore the history of software development to better understand the rationale and the problems that SmartCLIDE solves. Since the waterfall model was described in the early '70s, introducing a set of consecutive or linear steps for developing software (system and software requirements, analysis, design, coding, testing, and operations), several development paradigms have been described over the last 50 years. Primary evolution of waterfall was the V-life cycle, adopted by highly regulated sectors since it included a quality assurance layer that described a reverse waterfall process for verification and validation activities. When waterfall models were applied incrementally, we talked about incremental models. These models, though still linear, show the need to obtain an early functionality provision to obtain feedback and, therefore, try to reduce risk. For example, the Spiral Model added a risk analysis phase in each iteration. As another alternative to the rigid waterfall model, Rapid Application Development was proposed to deal with the flexibility of software development but required regular access to users. The Rational Unified Process (RUP) was the obtained result of a work that started looking into why software projects had failed and it went back to the spiral model. RUP divided the development process into four distinct phases each one involving business modelling, requirements, analysis and design, implementation, testing and deployment.

Agile + DevOps

However, despite all these attempts for more efficient software project management, until the late 90s many software projects of various sizes evolved into enormous disasters with huge budgets and time outruns. The need for a new paradigm as a response to waterfall models led to The Manifesto for Agile Software Development. It was a turning point in software development which brought together several of the values and principles already seen. The four values upon which the manifesto was signed are: (a) individuals and interactions over processes and tools; (b) working software over comprehensive documentation; (c) customer collaboration over contract negotiation; and (d) responding to change over following a plan.

In the agile way of work, software projects should be delivered incrementally, piece by piece where each piece is a fully functional unit of software. Software is being developed by small, robust, and self-organized teams which respond quickly and interact efficiently with the external environment. This is a smart way of working but requires a re-organization of many aspects of the software development lifecycle, infrastructure and deployment operations being one of them. If the team needs to build and deploy frequently and quickly every new functional software unit it needs to be able to do it with minimum interactions and communication overhead with other teams that are classically involved in these actions (network, database, infrastructure, middleware). This challenge led to the rise of a new area, the DevOps (Development Operations).

Since the first time the software business heard of DevOps in 2008, it has evolved really fast turning the buzzword into a reality that is transforming digital business all over the world. The philosophy behind DevOps aims at demolishing the walls that create operational silos in business, development and operations/infrastructures creating an environment where valuable work continuously flows, there is a continuous feedback up and downstream, and continuous improvement is a common practice. Among many other practices, the full autonomy and end-to-end responsibility of software development teams can be considered the cornerstone of DevOps. These practices mean that software creation teams will have the full responsibility to take an application to production: from the specification of requirements/user-stories to its deployment in a server. SmartCLIDE focuses on DevOps organizations offering assistance at all the stages of the software creation life cycle, namely: specification and planning, creation, verification, packaging, release, configuration and monitoring. The main practices that back autonomy and responsibility are the creation of multidisciplinary teams (including staff with business and operations knowledge), continuous communication, an extreme automatization of processes and the existence of a solid common knowledge base.

Microservices

Apart from the software operations area, Agile paradigm also transformed the software architecture scenery with the introduction of microservices. That’s because small, self-managed teams are more likely to develop small or medium-sized, autonomous, self-contained software modules which need a common execution platform along with rules for inter-module communication. This describes the microservices paradigm.

A microservices architecture is a development methodology wherein you can fragment a single application into a series of smaller services. Microservices are developed around business capabilities, and as such are independently deployable with automated deployment mechanisms. Related DevOps technologies can be used to help these automations. Each microservice is executing in its own process and interacting with lightweight mechanisms with other microservices or applications. This isolation and independence results in minimal management of these services, which are usually being built in different programming languages and employ different data storage technologies according to each element requirement. Below, we discuss the main features and benefits brought by microservice architectures.

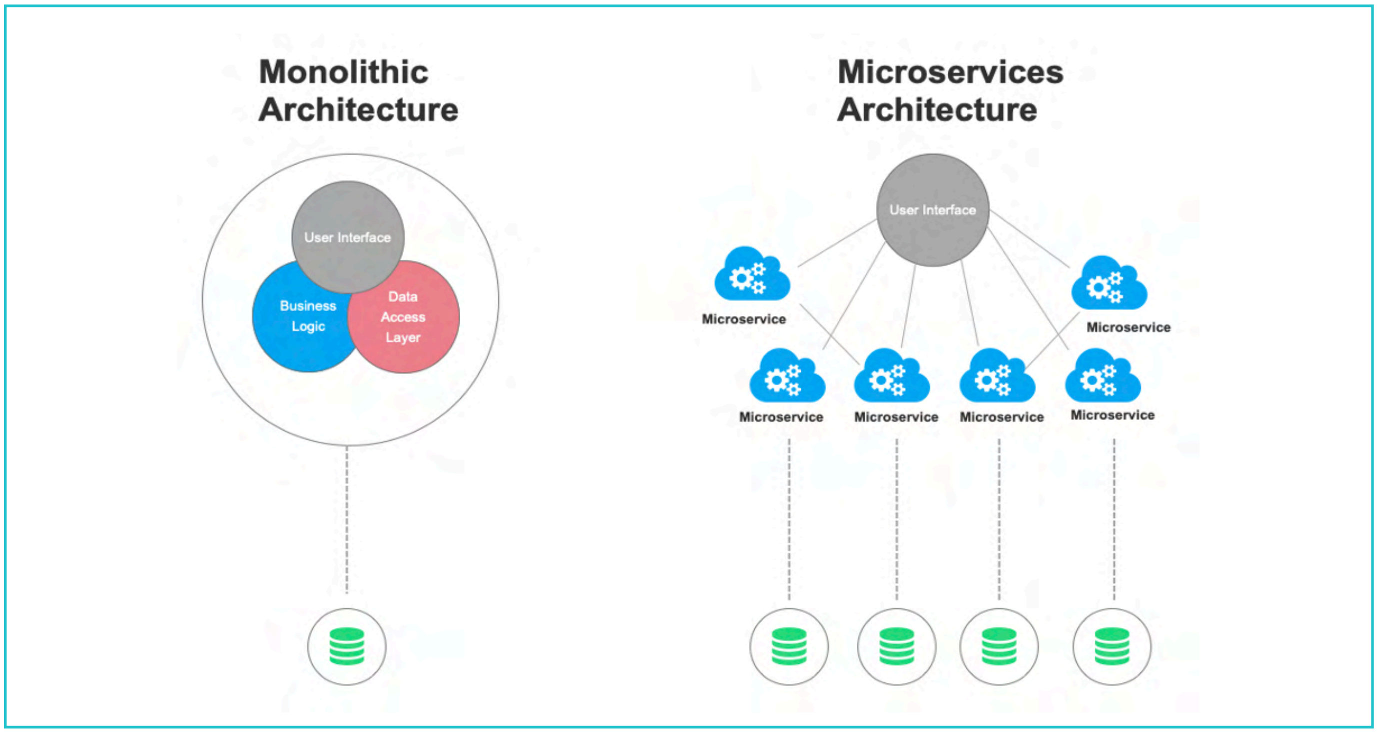


Figure 15: Monolithic vs. Microservices architecture

Dynamic Scalability. Based on the development of small isolated components, developer teams can easily scale up or down based on the requirements of a specific element. The flexibility of microservices lets a system expand fast without requiring a significant increase in resources. A monolithic architecture would require scaling the whole application. Each module in microservices can scale independently through: (a) X-axis scaling, by cloning with more memory or CPU; and (b) Z-axis scaling, by size using shading.

Technology Flexibility. This refers to the microservice architecture flexibility on its technology stack that leads to eliminating the constraints of vendor or technology lock-in and platform dependency. Each microservice can be built up using the software stack required for the specific element. Language, framework, data sources or any other dependencies required can be provided from a container without affecting the whole application design or the communication between the microservices in the ecosystem.

Easier and shorter development cycles. These are achieved through the important feature of agility that further leads to productivity and speed, smaller project development, ease of building and maintaining apps, that are independently DURS (deployed, updated, replaced & scaled). Since each microservice is a separate project, professionals can get involved in the process more easily because they do not have to study the system as a whole and they can work only on their part. Decomposing the monolithic structure into separate services, leads to team decomposition into more small engineering teams that work independently which increases agility. The modern Agile approach is tightly connected with practices as DevOps concepts, continuous integration (CI), and continuous deployment (CD). All of these practices allow for faster deployment, problem-solving, and time to market. This type of agility when combined with CI / CD tools, like Jenkins, and their underlying pipeline configuration capabilities, results in faster and smaller project development life cycle procedures. Compared to a microservices architecture, a monolithic architecture hampers the Agile and DevOps processes because of its tight connections between each and every component.

Fault Isolation. Small isolated microservices are less likely to affect the overall ecosystem when failing. A monolithic architecture is rigid when it comes to replacing functionalities or making changes. Small changes in one place can cause ripple effects, bugs and errors in the entire system due to the extreme coupling. As such microservices architecture improves replaceability and upgradeability of the system.

Reduced Downtime / Quick Response-time. Developers and DevOps could use another service when components fail, and the application continues its work independently. With the use of related technologies that provide virtual servers, containers, pods and clustering this architecture offer reduced response downtime.

The reader can find more details on this topic in the public deliverable D1.1-State-of-the-Art and Market Requirements.

Quality and security in a microservices world

Security Concerns in Microservices Architectures

Microservices are generally considered as a variant of service-oriented architecture and fortunately most aspects of security in microservice architecture are similar to monolithic applications. However, microservice architectural patterns introduce specific security challenges and problems, which should be treated differently. Based on the existing literature review and best practices adopted by many leading IT companies (e.g. Amazon, Netflix, Spotify, Twitter) we have identified several areas of security concerns and risk categories that have arisen along with the microservice paradigm.

An overview of security challenges in microservice architectures has been proposed in the form of a hierarchical decomposition in 6 layers: hardware, virtualization, cloud, communication, service/ application and orchestration.

Development of Secure Microservices

Microservices, as software products in nature, need to be developed having security in mind from the early stages of their development. Simply ensuring the implementation and deployment of mechanisms (either external or internal) that enhance the protection of the microservices with respect to important security aspects, including availability, confidentiality, and integrity, is not enough for fully protecting them against attacks.

Most of the software vulnerabilities stem from a small number of common programming errors. For instance, SQL injection and cross-site scripting, which are listed both by OWASP and NIST as the most dangerous and common vulnerabilities of web services and applications, are caused by lack of input validation/sanitization, which is a relatively simple programming error to address. Another source of security issues is the selection of insecure third-party reusable components and Application Programming Interfaces (APIs). Appropriate tooling is required to help them avoid the introduction of such security issues during the SDLC, and therefore write more secure code.

Automatic Static Analysis (ASA) tools have been proven effective in uncovering security-related bugs early enough in the software development process. They are applied directly to the source or compiled code of the system, without requiring its execution. In fact, automatic static analysis is considered an important technique for adding security during the software development process. Moreover, static analysis is believed to be more effective in detecting code-level vulnerabilities compared to other dynamic testing approaches like penetration testing and fuzzing, as it is observed to produce significantly fewer false negatives.

Quality Assessment through Machine Learning

Machine Learning technologies have been applied to resolve multiple and quite diverse research problems such as defect management, cost/effort estimation, management of design-time quality attributes, recommendations for efficient project management, and detection of security threats. In terms of quality attributes, the most relevant ones appear to be the improvement of maintainability and functional suitability (i.e., correctness), followed by security and business quality attributes. Overall, the following practices can be mapped to quality management:

Cost/Effort Estimation: Monetization is a key concept in quality management. To this end, any cost or effort estimation approach based on past data can be considered as relevant to predict the cost of applying refactoring or to predict the cost of future maintenance effort. In this category of Software Engineering problems special emphasis is placed on studies that deal with software maintenance effort prediction.

Management of Design-Time QAs and Defects: In this high-level category, various related problems have been identified. First, many studies focused on change- and fault-proneness. These concepts are closely related to interest probability, in the sense that changes and faults lead to maintenance activities that can accumulate interest. Additionally, other studies focus on the detection of small occurrences. Finally, any method that is used for assessing or characterizing the levels of QAs (e.g., maintainability) can be useful.

Requirements and Project Management Recommendations: Many studies use ML to provide recommendations to developers related to which requirements shall be implemented first, or which reported bugs shall be prioritized. Such recommendations could be useful for Technical Debt prioritization, by considering that artefacts that are not expected to change (due to bug fixing, or implementation of new requirements), shall not be prioritized for design-time quality improvements.

Given the above, we can conclude to the following baseline market requirements for the development of the SmartCLIDE solution:

SmartCLIDE shall support	SmartCLIDE should support	SmartCLIDE may support
1. user-friendly GUI even for non-technical users 2. visually intuitive interfaces to help users with model generation and training 3. implementation of coding-by-example principle 4. the provision of abstractions to minimize manual intervention that are required by the developers to the source code for implementing new features 5. the classification of services, learning from code or applying Machine Learning algorithms 6. user stories, features specification 7. specification of acceptance criteria for functional and non-functional requirements 8. the short iterations concept 9. Continuous Integration / Continuous Development	21. refactoring 22. easy configuration 23. the provision of metrics for maintainability / reusability at the service and the system level 24. the extension of existing tools for measuring maintainability and reusability to capture the metrics at the service level 25. the provision of solutions for facilitating the identification and elimination of critical vulnerabilities that reside in the source code of microservices from the early stages of their development 26. the provision of an easy non-coding implementation for DL usage (general problems) depending on input data 27. the provision of code blueprints (skeletons) based on Gherkin inputs for services implementation 28. the discovery and	35. generation of automatic tests by natural language interpretation of acceptance tests 36. the provision of on-the-fly suggestions on how to improve the reusability and maintainability of the system 37. SmartCLIDE may support 38. agile tools such as a Kanban board 39. implementation of artefacts for product and sprint backlog management (e.g.

(CI/CD) 10. automated testing in differentflavours: Acceptance Test-Driven Development (ATDD) / Behavioral Driven Development (BDD) / Test-Driven Development (TDD). 11. static analysis 12. working code as a source of documentation 13. integration with run-time monitoring tools 14. version control of software 15. cloud native IDE for cloud native solutions 16. Business Process Modelling capabilities 17. service discovery and search 18. service integration through the online dashboard 19. a wrapper which isolates user from Deep Learning (DL) complexity as far as possible, releasing developers from boilerplate code generation 20. the provision of coding blueprints which can serve as a base for more complex tasks, making code more reusable and easier to understand	composition of basic services based on ontologies 29. scalability of processing capability 30. replicability of architecture to increase flexibility 31. fault tolerance and reliable 32. security through isolation / dependability 33. the monitoring of maintainability and reusability of the project under development 34. dynamic software configuration	Kanban or Scrumban boards) 40. implementation of artefacts facilitating waterfall life cycles

SmartCLIDE Service Creation
By University of Macedonia

SmartCLIDE Service Creation aims to deliver a faster, easier, and more user-friendly solution than from-scratch development of services. The approach aims to automate several steps and easily provide functionalities to developers, in a manner that enhances productivity and minimizes distractions and time-consuming interactions with external tools. SmartCLIDE analyzes the required steps a developer has to perform so as to deliver a fully functional service, in order to come up with ways to fine-tune the flow.

Before starting the development process, several steps are required to setup the development process:

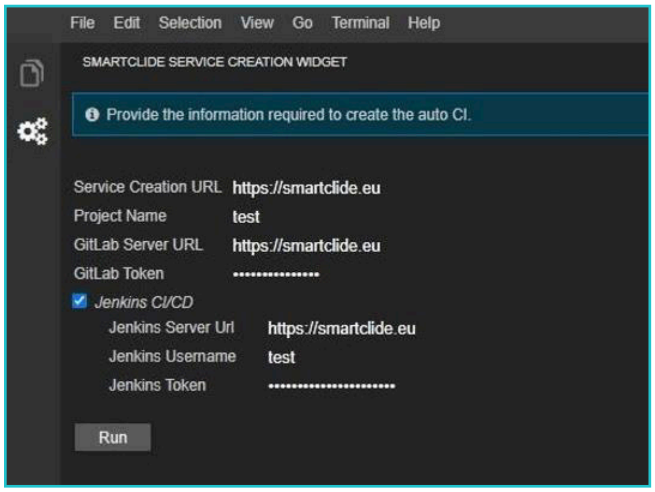
- Creating a code repository, for e.g., a GitLab project
- Creating a CI/CD pipeline, for e.g., GitLab CI/CD or a Jenkins job
- Configuring them both to interact with each other

During code development, e.g., for the creation of a new service the developer needs to interact with several tools:

- for writing code
- for collaborative development
- for performing tests and verifying the results
- for evaluating the quality and maintainability of the code
- for monitoring the progress of the project
- making the service invokable by being packaged into an image

Considering the actual coding part as the central point of software development, the main UI for service creation should be the IDE. Several code development tools were evaluated, and Eclipse Theia was selected as the final solution. Following our previous way of thinking, all the other service creation functionalities should be accessible through the **Eclipse Theia IDE** to minimize distractions and make the whole process easier and more efficient.

Starting with the creation of the necessary structure for development, a new widget was developed for the IDE.



SMARTCLIDE SERVICE CREATION WIDGET

Provide the information required to create the auto CI.

Service Creation URL	https://smartclide.eu
Project Name	test
GitLab Server URL	https://smartclide.eu
GitLab Token	••••••••••••••

[✓] *Jenkins CI/CD*

Jenkins Server Url	https://smartclide.eu
Jenkins Username	test
Jenkins Token	•••••••••••••••••••••

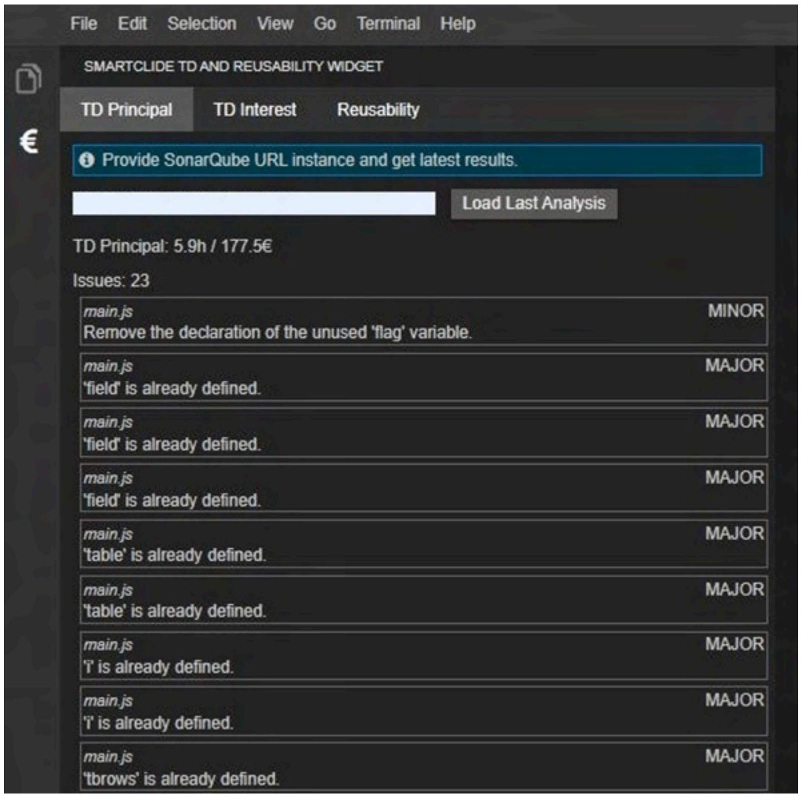
Figure 16: Service Creation Widget

Here the user can input the required information, and the Service Creation component will create and configure a new GitLab project. For now, there is also the ability to use Jenkins as the CI/CD server. In that case, the GitLab project and the Jenkins job pairing is done by the component automatically in the background, thus relieving the user from an otherwise dull and time-consuming sequence of steps. After the completion of the structure creation, the user is presented with a new Git URL.

Cloning the Git repository using Theia's Git commands, the user can begin to code. Through Theia's integrated Git, the process of staging, committing, and pushing code changes is made easy. Considering that the code repository is either paired with a Jenkins job or handled by GitLab CI/CD, every code push triggers a pipeline. During the pipeline execution, tests and code analysis can be executed automatically.

Provides valuable information to the user, is the evaluation of the quality and maintainability of the code.

The next step, that provides valuable information to the user, is the **evaluation of the quality and maintainability** of the code. This is achieved by using another newly created widget that acts as a middleman between the IDE and the SmartCLIDE backend services.



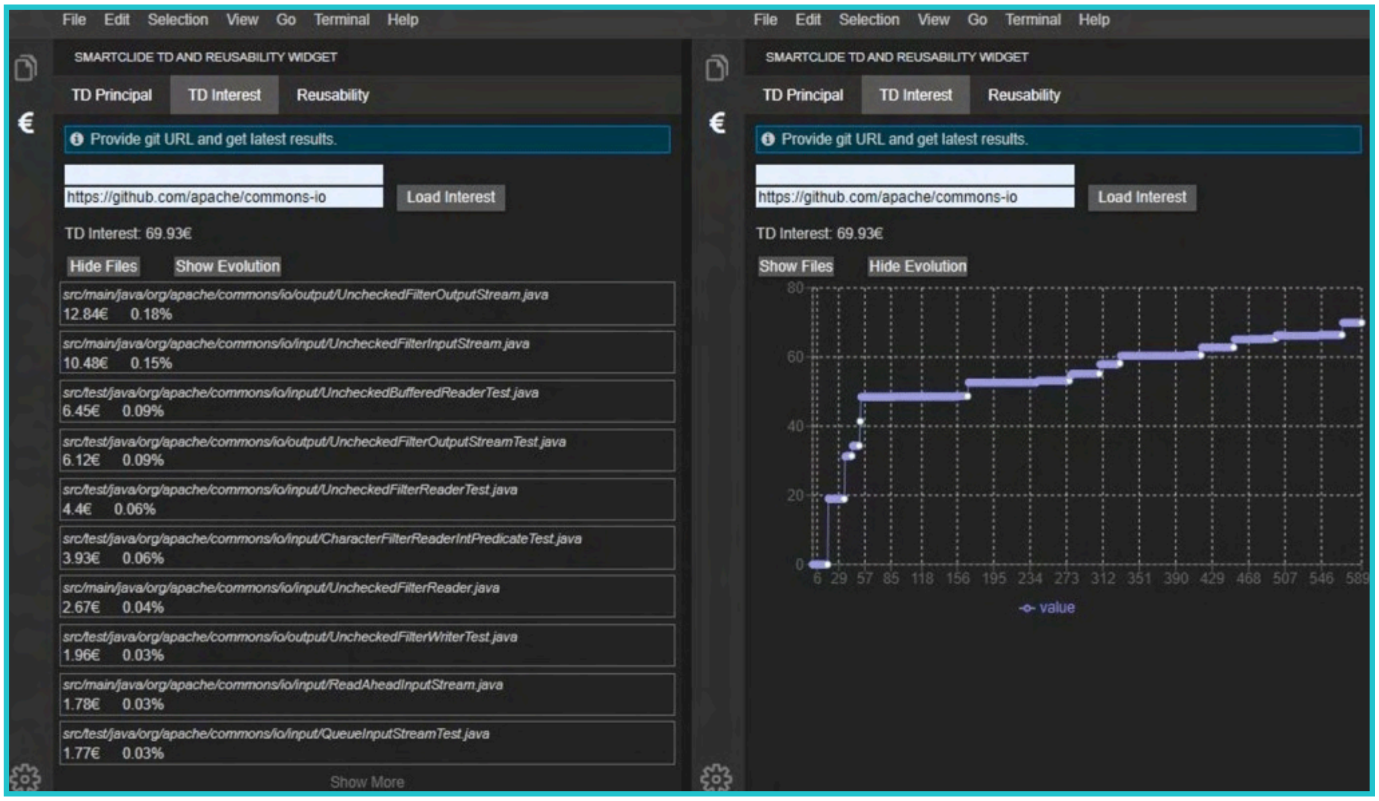
Provide SonarQube URL instance and get latest results.

TD Principal: 5.9h / 177.5€

Issues: 23

main.js Remove the declaration of the unused 'flag' variable.	MINOR
main.js 'field' is already defined.	MAJOR
main.js 'field' is already defined.	MAJOR
main.js 'field' is already defined.	MAJOR
main.js 'table' is already defined.	MAJOR
main.js 'table' is already defined.	MAJOR
main.js 'i' is already defined.	MAJOR
main.js 'i' is already defined.	MAJOR
main.js 'tbrows' is already defined.	MAJOR

Figure 17: Technical Debt Principal widget



Provide git URL and get latest results.

https://github.com/apache/commons-io

TD Interest: 69.93€

src/main/java/org/apache/commons/io/output/UncheckedFilterOutputStream.java
12.84€ 0.18%

src/main/java/org/apache/commons/io/input/UncheckedFilterInputStream.java
10.48€ 0.15%

src/test/java/org/apache/commons/io/input/UncheckedBufferedReaderTest.java
6.45€ 0.09%

src/test/java/org/apache/commons/io/output/UncheckedFilterOutputStreamTest.java
6.12€ 0.09%

src/test/java/org/apache/commons/io/input/UncheckedFilterReaderTest.java
4.4€ 0.06%

src/test/java/org/apache/commons/io/input/CharacterFilterReaderIntPredicateTest.java
3.93€ 0.06%

src/main/java/org/apache/commons/io/input/UncheckedFilterReader.java
2.67€ 0.04%

src/test/java/org/apache/commons/io/output/UncheckedFilterWriterTest.java
1.96€ 0.03%

src/main/java/org/apache/commons/io/input/ReadAheadInputStream.java
1.78€ 0.03%

src/test/java/org/apache/commons/io/input/QueueInputStreamTest.java
1.77€ 0.03%

Provide git URL and get latest results.

https://github.com/apache/commons-io

TD Interest: 69.93€

Figure 18: Technical Debt Interest widget (per File and Evolution)

The user provides the required information, in this case, the Git URL of the project, and receives the results of code analysis. Depending on the user-specified steps of the pipeline, the analysis process can vary in execution time. The request is passed to the SmartCLIDE backend services, where the **Reusability Index, Technical Debt Principal, and Technical Debt Interest** are calculated for the project.

As a final step, again using the pipeline, you have the ability to package the finished project to a Docker image ready for deployment. Of course, considering that the above are part of an early prototype, there is pending research and experimentation in order to find an optimum and user-friendly approach.

SmartCLIDE Deep Learning Engine

By AIR Institute

The rapid rate of technological and digital advancement requires the building of related software, which is a time-consuming process. SmartCLIDE includes the advantages of Artificial Intelligence (AI) and Cloud Computing. These technologies can help the developer overcome the complexities associated with multi-platform software products.

Merging artificial intelligence with existing IDE functionality can bring new opportunities in the most involved area in software development tools. This improvement can include improving current IDE features, such as code suggestion or code search, resorting to recommender systems to provide more accurate results to developers. Moreover, With the advent of online code repositories and improved data collection, it has become possible to add more intelligent functionalities in most of the automation tasks, such as service classification.

Concerning Intelligent software engineering, theoretical [1][2] and empirical [3][4] works have shown that software intelligence has been widely used in software development. Accordingly, SmartCLIDE has proposed the DLE component, which is responsible for feeding smart Assistants by intelligent models. DLE subcomponent responsibility can fall into the following categories:

- Context monitoring specification in order to provide suggestions
- AI code completion for generating one-line code using language modelling The acceptance test set suggestion for giving the user a set of tests defined in Gherkin format.
- Classification of Web services based on their meta-data in order to reduce service selection search space
- Code repository suggestion is responsible for making suggestions for the user to facilitate commits to the git repository.
- Service deployment environment recommendations in order to produce suggestions for the sizing of the deployment environment.
- BPMN Items suggestions aim to help automation in selecting the next node/item in the BPMN workflow

Component name	Description
Service Classification Model	The Service classification sub-component is responsible for classifying new services. There are two important resources for new services. First, the newly created services are created by SmartCLIDE users using the service creation module. Second, the new observation by service discovery module.
Template-based agent Code Generation	This subcomponent is responsible for generating code based on internal templates. The API returns related code snippets based on templates to implement the workflow represented in BPMN in low code. The first version of this API is designed for finding Java codes.
Code Generation Auto-complete Model	This sub-component is responsible for one line automatic code generation based on DL learning model, which is trained by available public source codes.
Code Repository Suggestions Model	This wizard is responsible for generating suggestions to the user to facilitate commits to the git repository. Receiving information from the monitoring system, and with the help of the DLE, it will determine the best time to commit to the git repository.
Deployment environment suggestions Model	This sub-component is responsible for generating suggestions for the sizing of the deployment environment
Acceptance test Suggestions Model	The acceptance test set suggestion system, based on collaborative filtering techniques, is responsible for providing the user with a set of tests defined in Gherkin format to be applied to the workflow defined in the BPMN and help verify if the expectations are met
BPMN Items suggestions	The BPMN Items suggestion system consists of automatically selecting the next node/item in the workflow being modeled during service composition in BPMN format.

Predictive Model tool API	This wizard, as a subcomponent of the DLE in SmartCLIDE, is accessible through a RESTful API in several stages, structured in an ideally linear flow that in practice allows iterative backtracking. Its objective is to guide the user in the creation of a predictive model.

If you wish to learn more about this aspect of the SmartCLIDE project, we invite you to read the public deliverable entitled "D3.1 – Early SmartCLIDE Cloud IDE Design".

References

[1] "JAXEnter, What Theia is all about." [Online].
Available: https://jaxenter.com/theia-ide-efftinge-interview-134467.html

[2] "What are Message Brokers?," Aug. 11, 2021.
https://www.ibm.com/cloud/learn/message-brokers (accessed Sep. 04, 2021).

[3] "What is Usability Testing?," The Interaction Design Foundation.
https://www.interaction-design.org/literature/topics/usability-testing (accessed Sep. 07, 2021).

[4] "JUnit 5." https://junit.org/junit5/ (accessed Sep. 07, 2021).

SmartCLIDE User Interface

By UNPARALLEL

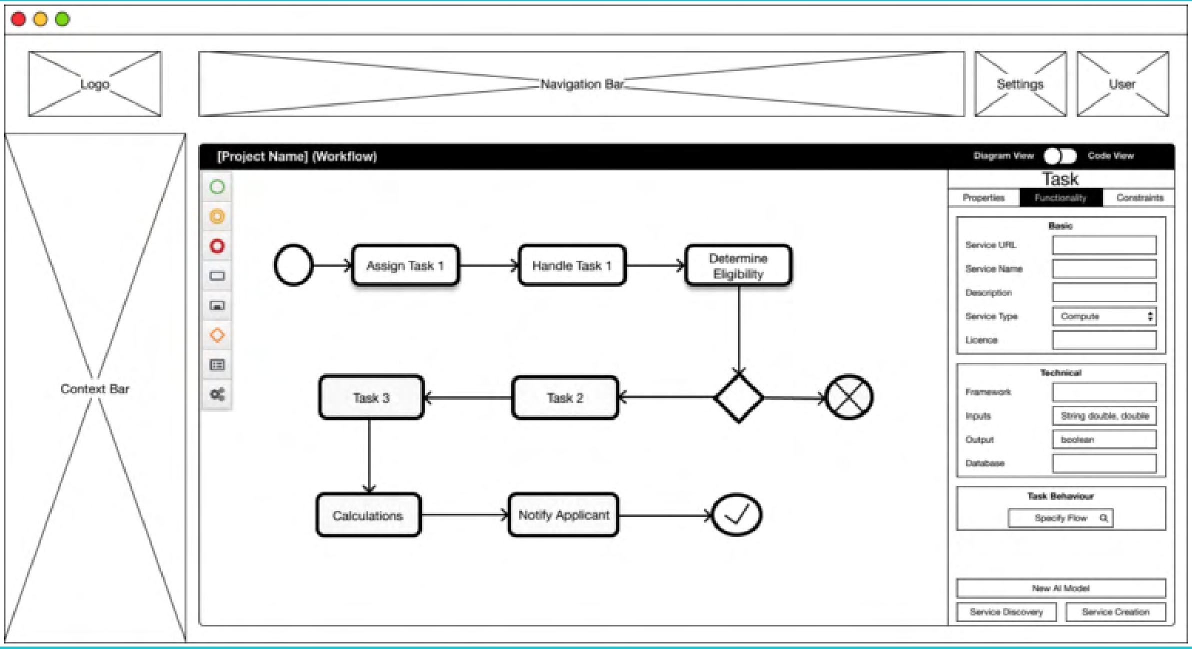
This article describes the design and current development progress of the three main components of the User Interface: Toolbox, Workbench, and Run-time Simulation & Monitoring Console.

The front-end user interface was designed considering the three main concepts in which SmartCLIDE's functionalities can be grouped:

- Workflows – result from combining services using Business Process Model andNotation (BPMN) diagrams.
- Services – resources available through an URL that can be integrated to createcomplex scenarios.
- Deployments – instances of services or workflows that run on specific environments(e.g., Amazon Web Services).

Workflows

The actual design of the workflow requires a BPMN editor. The elements are dragged onto the drawing area and the fields from the "Properties" and "Functionality" tabs of each node/task must be completed. Throughout this process, the Smart Assistant aids the developer by suggesting nodes as the workflow is being designed.



Service URL	
Service Name	
Description	
Service Type	Compute
Licence	
Framework	
Inputs	String double, double
Output	boolean
Database	

Figure 19: BPMN Editor

At any time, the user can change to the "Code Editor" tab, inspect, and manually edit the code being generated by SmartCLIDE using an instance of the Theia code editor

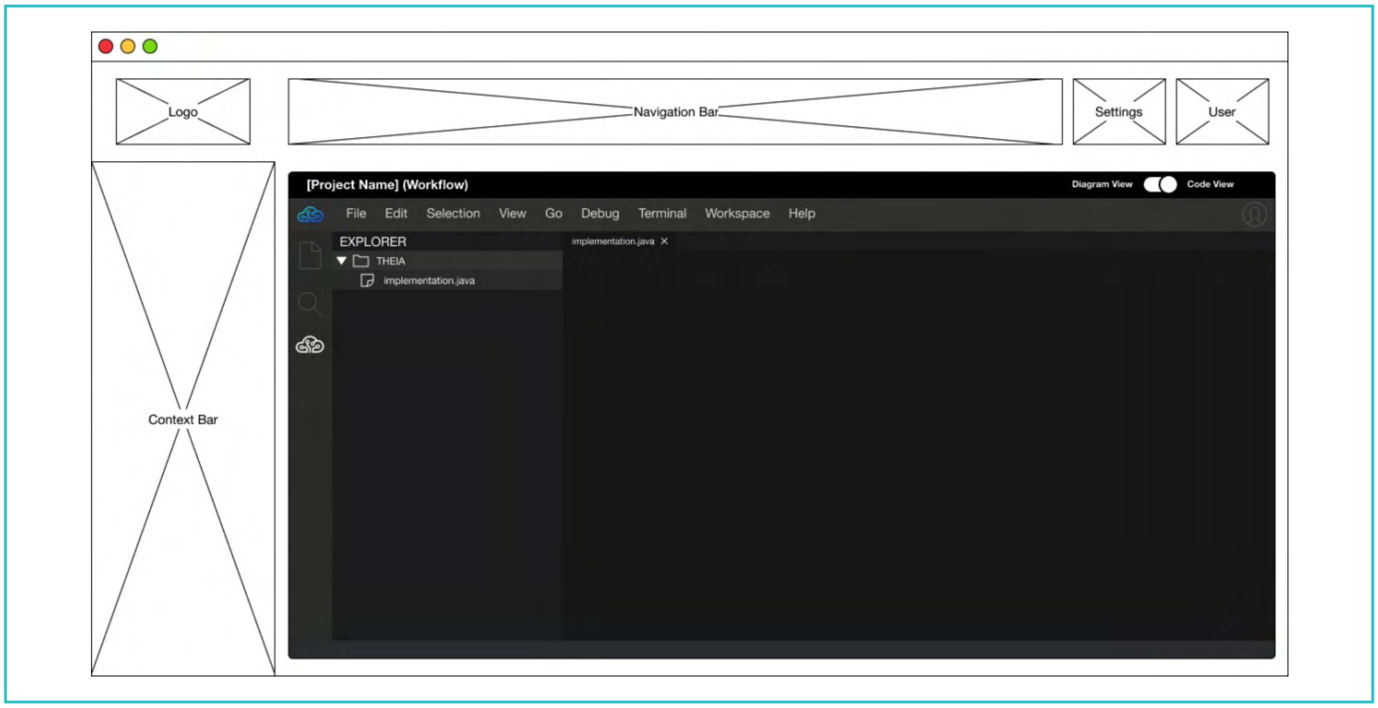
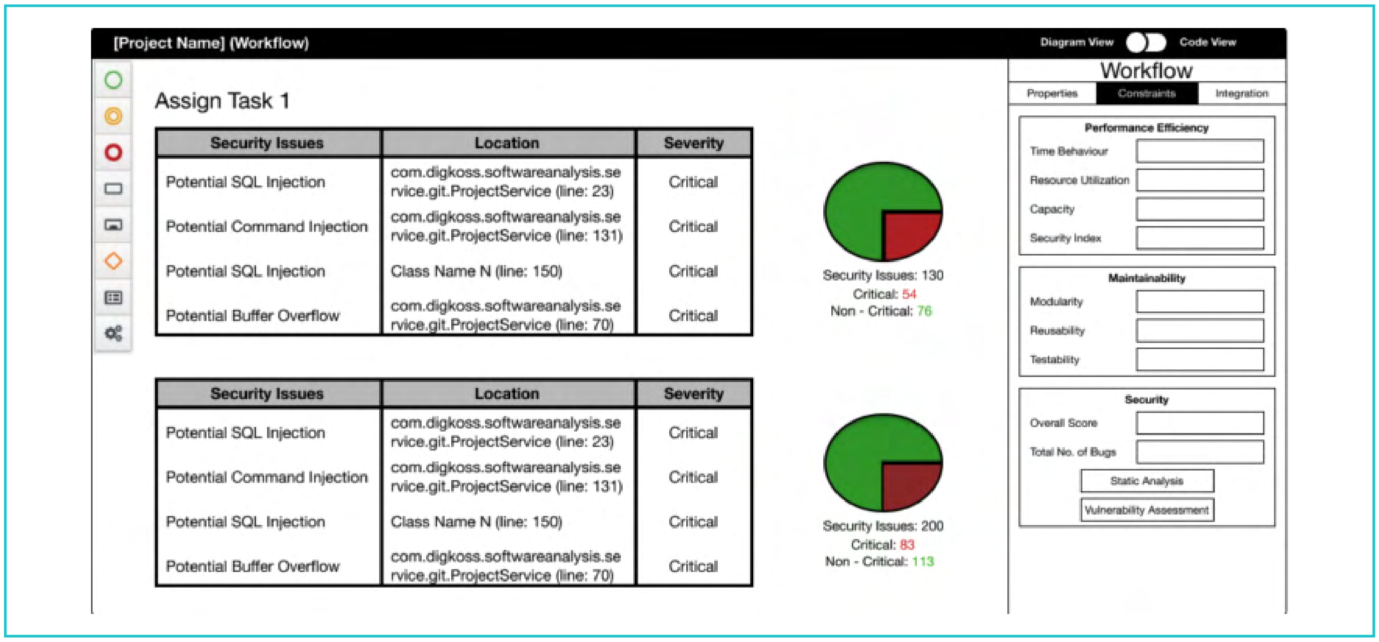


Figure 20: Eclipse Theia code editor

A task can be easily implemented using an existing service. For that, SmartCLIDE provides the Service Discovery tool which receives details of new registries, analyses them, and suggests services that match the meta-data provided by the developer. The default usages include deployable versions, services connected to the web or services in source code.



Security Issues	Location	Severity
Potential SQL Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 23)	Critical
Potential Command Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 131)	Critical
Potential SQL Injection	Class Name N (line: 150)	Critical
Potential Buffer Overflow	com.digkoss.softwareanalysis.service.git.ProjectService (line: 70)	Critical

Security Issues	Location	Severity
Potential SQL Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 23)	Critical
Potential Command Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 131)	Critical
Potential SQL Injection	Class Name N (line: 150)	Critical
Potential Buffer Overflow	com.digkoss.softwareanalysis.service.git.ProjectService (line: 70)	Critical

Figure 21: Security analysis page

When the workflow is completed, the developer can test it.

SmartCLIDE allows the user to run in the background a security analysis on the workflow, as well as assessing its vulnerability, namely identifying potential security breaches.

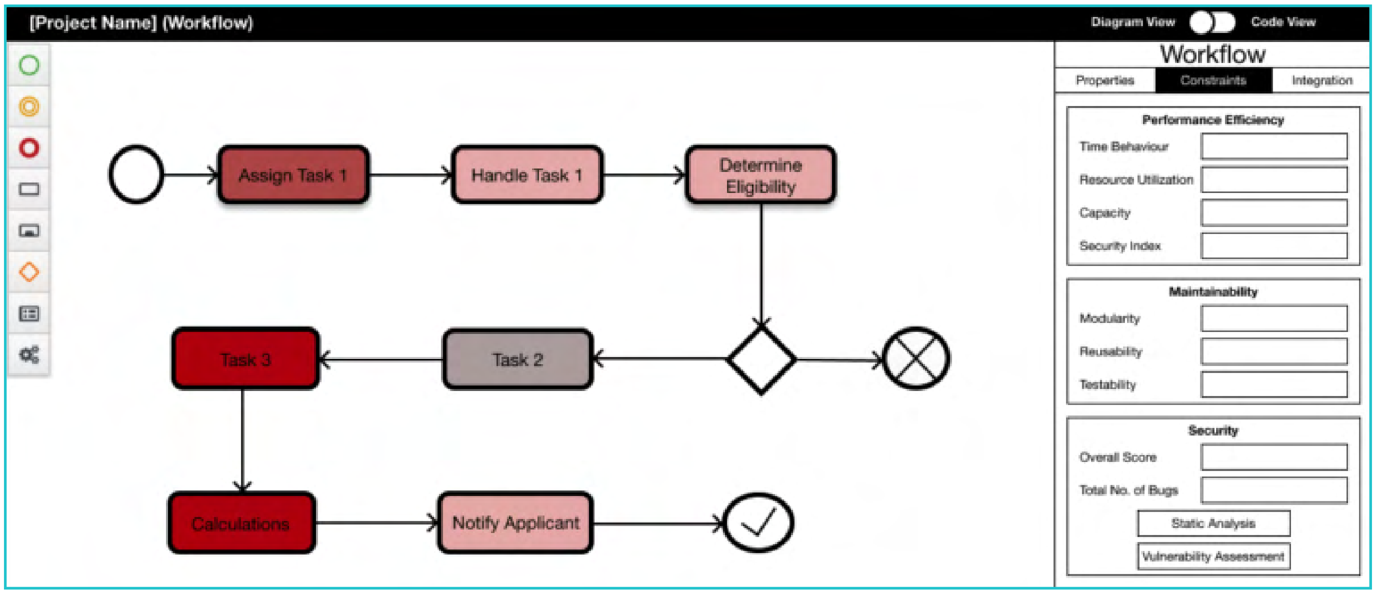
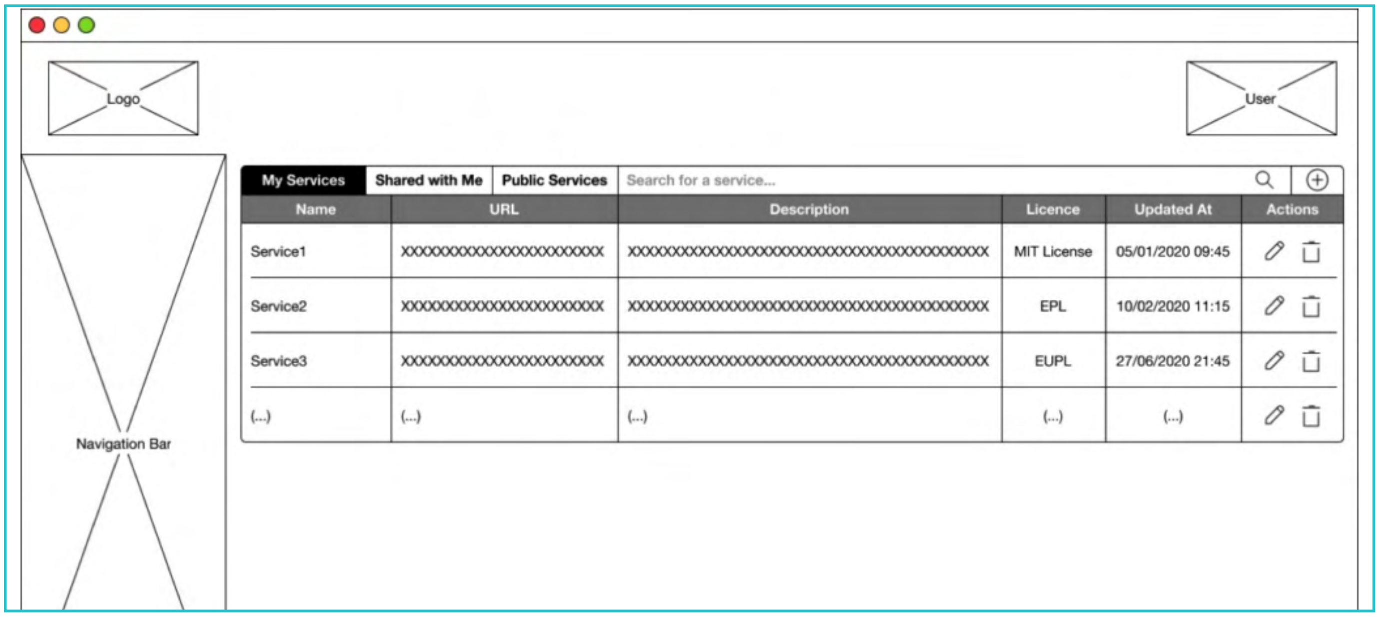


Figure 22: Vulnerability assessment page

Services

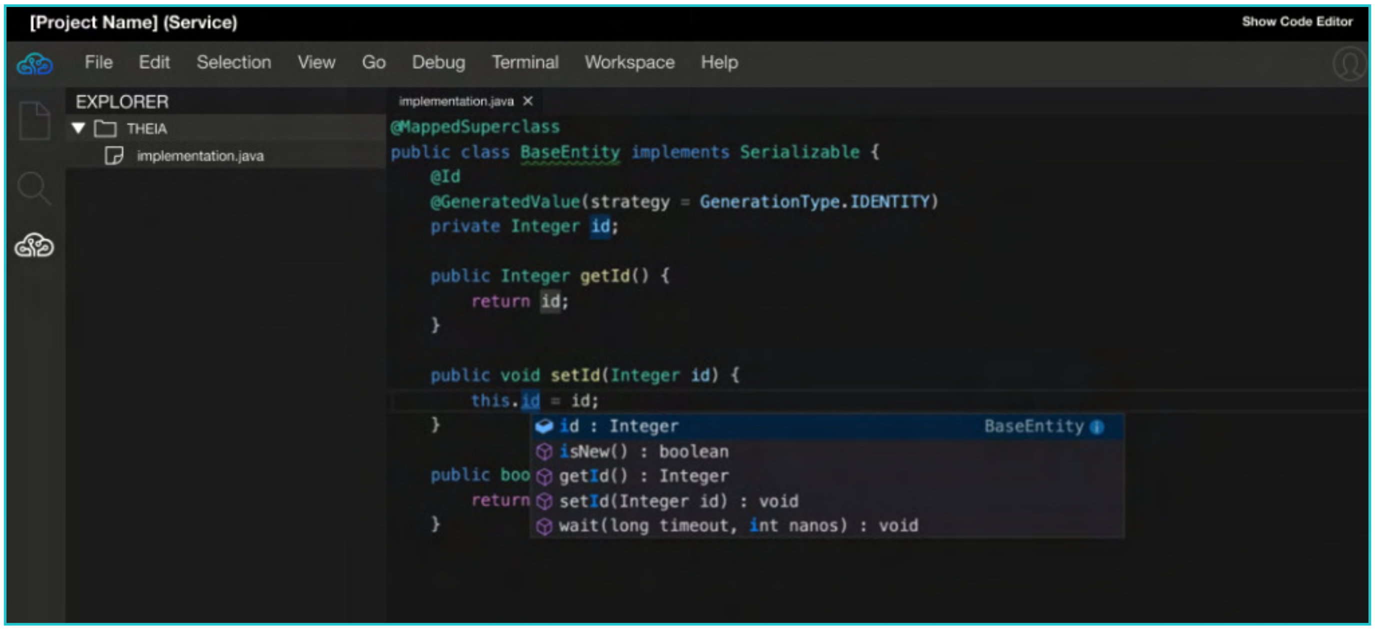
As in the case of the workflows, in the Services page, the user can filter the services by developer ("My Services", "Shared with Me" or "Public Services") or any keyword (i.e., name, URL, description, or license) or value (update date) written in the search bar. Finally, this page is the starting point for creating, editing, or removing services as well.



Name	URL	Description	Licence	Updated At	Actions
Service1	xxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	MIT License	05/01/2020 09:45	
Service2	xxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	EPL	10/02/2020 11:15	
Service3	xxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	EUPL	27/06/2020 21:45	
(...)	(...)	(...)	(...)	(...)	

Figure 23: The main page of the services

SmartCLIDE also provides an Eclipse Theia instance, as a source code editor. For the services' implementation, the Smart Assistant helps the developer with code auto-completion, .../...



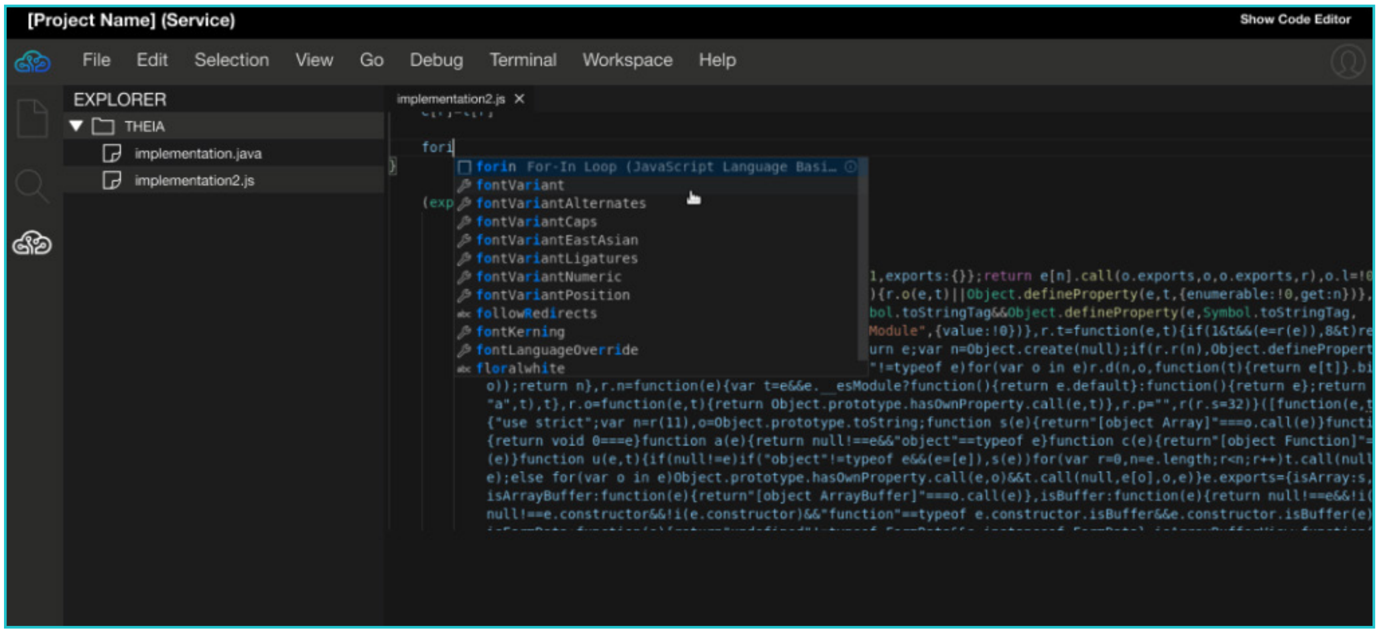
```
@MappedSuperclass
public class BaseEntity implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }           id : Integer                          BaseEntity
                isNew() : boolean
    public boo  getId() : Integer
        return  setId(Integer id) : void
    }           wait(long timeout, int nanos) : void
```

Figure 24: Code auto-completion

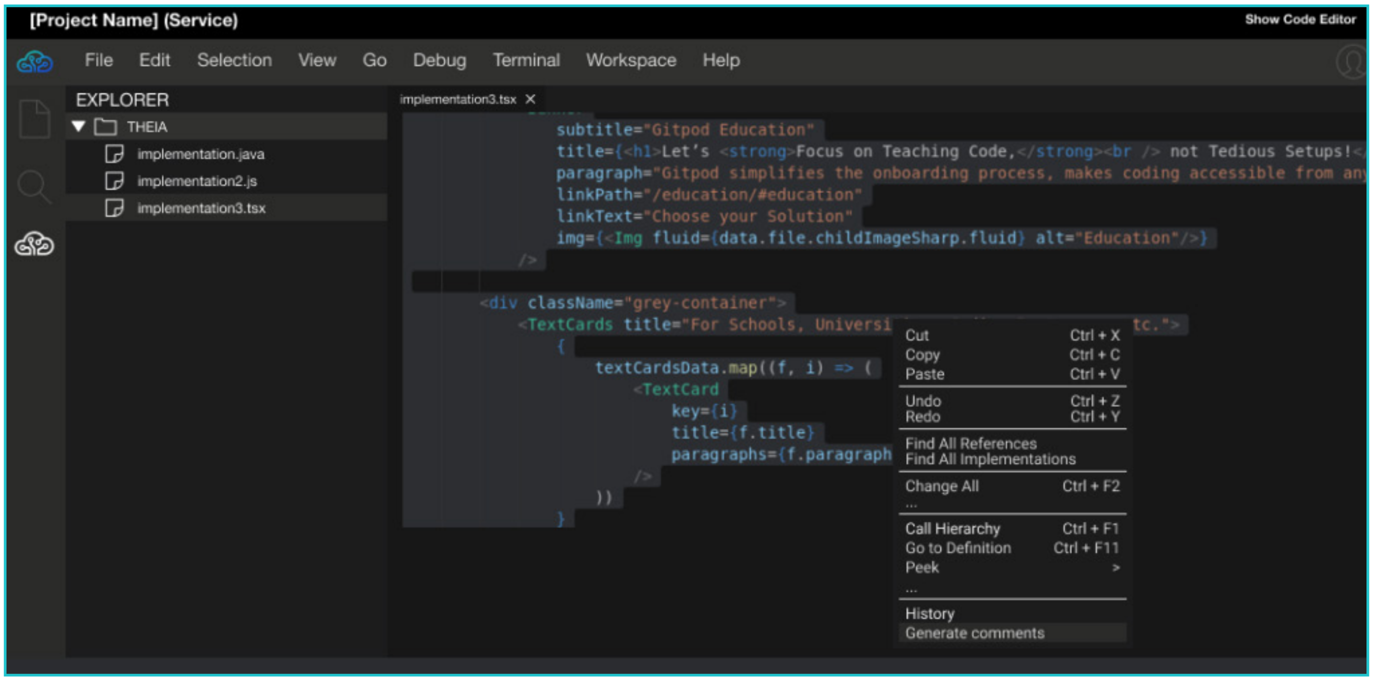
.../... live template recommendations, .../...



```
    fori
}       forin For-In Loop (JavaScript Language Basi…
        fontVariant
    (exp fontVariantAlternates
        fontVariantCaps
        fontVariantEastAsian
        fontVariantLigatures
        fontVariantNumeric                 1,exports:{}};return e[n].call(o.exports,o,o.exports,r),o.l=!0
        fontVariantPosition                ){r.o(e,t)||Object.defineProperty(e,t,{enumerable:!0,get:n})},
        followRedirects                    bol.toStringTag&&Object.defineProperty(e,Symbol.toStringTag,
        fontKerning                        Module",{value:!0})},r.t=function(e,t){if(1&t&&(e=r(e)),8&t)re
        fontLanguageOverride               urn e;var n=Object.create(null);if(r.r(n),Object.definePropert
        floralwhite                        "!=typeof e)for(var o in e)r.d(n,o,function(t){return e[t]}.bi
        o));return n},r.n=function(e){var t=e&&e.__esModule?function(){return e.default}:function(){return e};return
        "a",t),t},r.o=function(e,t){return Object.prototype.hasOwnProperty.call(e,t)},r.p="",r(r.s=32)}([function(e,t
        {"use strict";var n=r(11),o=Object.prototype.toString;function s(e){return"[object Array]"===o.call(e)}functi
        {return void 0===e}function a(e){return null!==e&&"object"==typeof e}function c(e){return"[object Function]"=
        (e)}function u(e,t){if(null!=e)if("object"!=typeof e&&(e=[e]),s(e))for(var r=0,n=e.length;r<n;r++)t.call(null
        e);else for(var o in e)Object.prototype.hasOwnProperty.call(e,o)&&t.call(null,e[o],o,e)}e.exports={isArray:s,
        isArrayBuffer:function(e){return"[object ArrayBuffer]"===o.call(e)},isBuffer:function(e){return null!==e&&!i(
        null!==e.constructor&&!i(e.constructor)&&"function"==typeof e.constructor.isBuffer&&e.constructor.isBuffer(e)
```

Figure 25: Live template recommendation

.../... comments generation (and service testing automation (using JUnit).



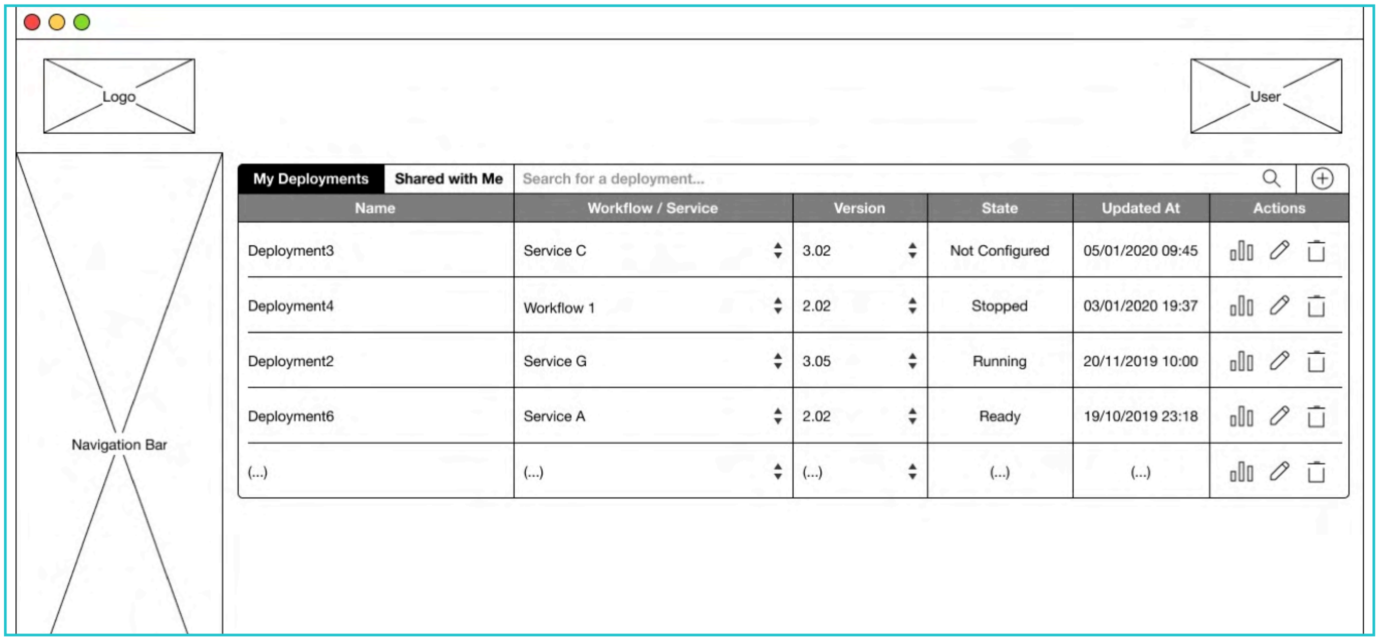
```
            subtitle="Gitpod Education"
            title={<h1>Let's <strong>Focus on Teaching Code,</strong><br /> not Tedious Setups!<
            paragraph="Gitpod simplifies the onboarding process, makes coding accessible from any
            linkPath="/education/#education"
            linkText="Choose your Solution"
            img={<Img fluid={data.file.childImageSharp.fluid} alt="Education"/>}
        />

    <div className="grey-container">
        <TextCards title="For Schools, Universi                              tc.">
            {
                textCardsData.map((f, i) => (
                    <TextCard
                        key={i}
                        title={f.title}
                        paragraphs={f.paragraph
                    />
                ))
            }
```

Figure 26: Comments generation

Deployments

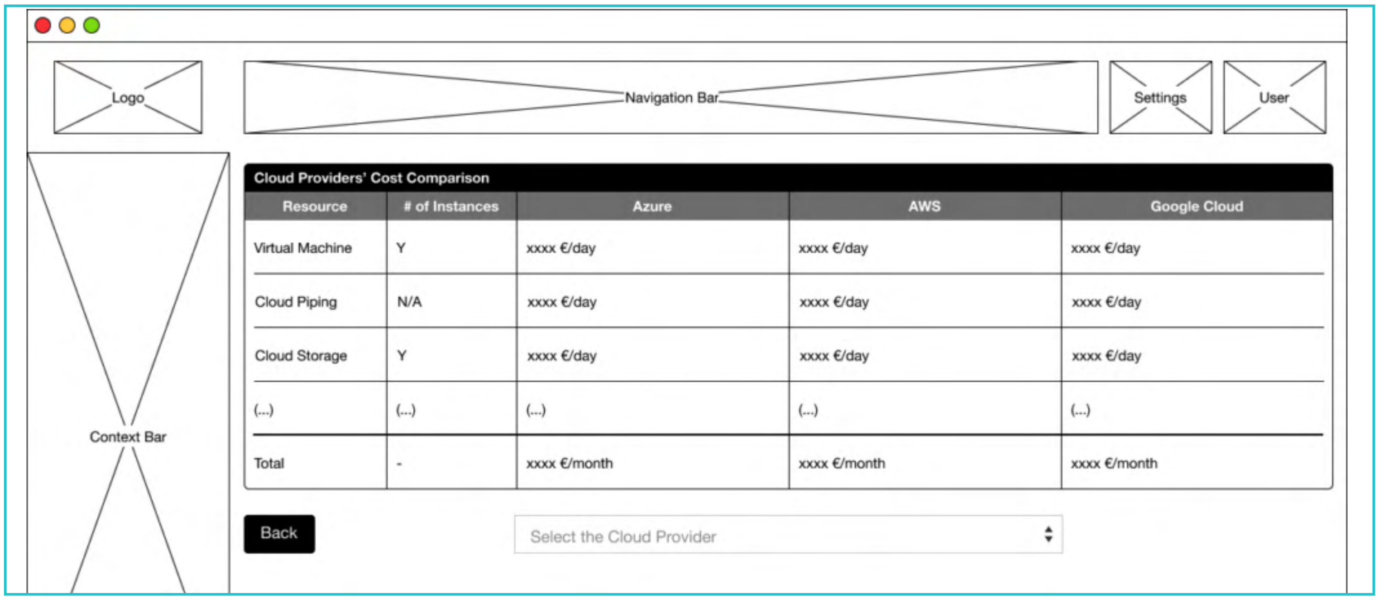
On the Deployments page, the user can monitor his own deployments and deployments shared with him. In addition, through this page, the user can create new and edit or remove past deployments.



Name	Workflow / Service	Version	State	Updated At	Actions
Deployment3	Service C	3.02	Not Configured	05/01/2020 09:45	
Deployment4	Workflow 1	2.02	Stopped	03/01/2020 19:37	
Deployment2	Service G	3.05	Running	20/11/2019 10:00	
Deployment6	Service A	2.02	Ready	19/10/2019 23:18	
(...)	(...)	(...)	(...)	(...)	

Figure 27: Main page of the deployments

Before deploying the workflow/service, the user can see a cost comparison that assists in choosing the best cloud provider and then, go back to the deployment configuration page. It is worth mentioning that the cost simulation service will only be able to provide accurate values at the component level.



Cloud Providers' Cost Comparison

Resource	# of Instances	Azure	AWS	Google Cloud
Virtual Machine	Y	xxxx €/day	xxxx €/day	xxxx €/day
Cloud Piping	N/A	xxxx €/day	xxxx €/day	xxxx €/day
Cloud Storage	Y	xxxx €/day	xxxx €/day	xxxx €/day
(...)	(...)	(...)	(...)	(...)
Total	-	xxxx €/month	xxxx €/month	xxxx €/month

Figure 28: Main page of the cost comparison service

The user can choose the metrics to be automatically monitored during runtime in the deployment configuration page. From the main page of deployments, the user monitors the selected metrics. The data of the metrics are collected by the Runtime Monitoring & Verification and the Context Handling components.

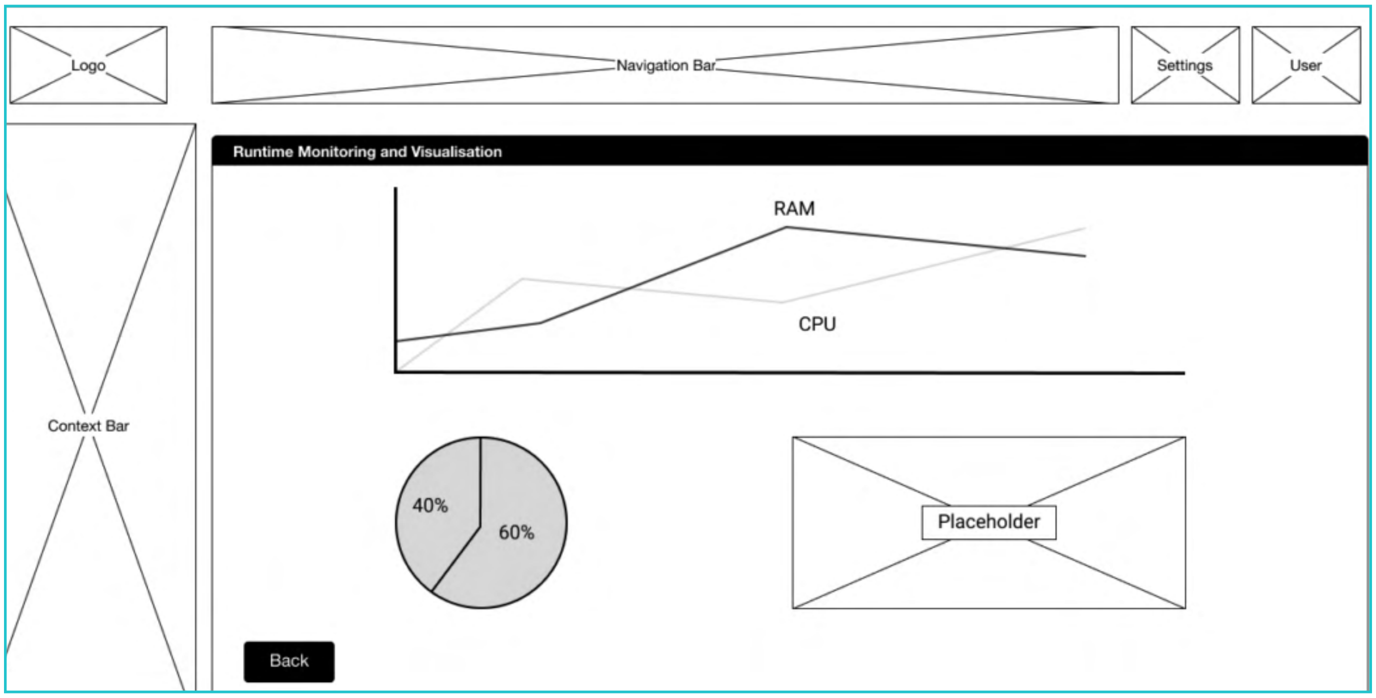


Figure 29: Runtime metrics monitoring and visualization page

If you wish to learn more about this aspect of the SmartCLIDE project, we invite you to read the public deliverable entitled "D3.1 – Early SmartCLIDE Cloud IDE Design".

SmartCLIDE DLE Component

By AIR Institute

The interest in building software is increasing with the move towards online businesses. However, this process demands the building of customised software for the target business, which is time-consuming. Accordingly, several models and concepts have emerged to create software faster. One of the major topics is software reuse, which is the process of utilising existing components to build new software. By increasing online services in public resources and service registries, **software reuse** has captured the attention of engineers. These online services include a wide range of software, applications, or cloud technologies that use standard protocols to communicate and exchange data messaging. Moreover, developer task automation can improve composing available online services. Automation includes concepts and techniques that apply to developer tasks to increase productivity and reduce human errors.

In this context, the SmartCLIDE toolkit tries to **bring most service development tasks together** and also add automation techniques. This automation includes **rule-based or AI-based approaches,** which are presented as functionality to help developers.

These functionalities allow developers to invest much more time into their domain problem and software business logic rather than manual proper service identification and development.

This article aims to introduce SmartCLIDE DLE models, a subset of intelligent software development that refers to applying intelligent techniques to software development. Proposed models try to provide a learning algorithm with the information that data carries. SmartCLIDE data include internal data history and external online web services identified from online resources. The following figure demonstrated the big picture of SmartCLIDE external service identification.

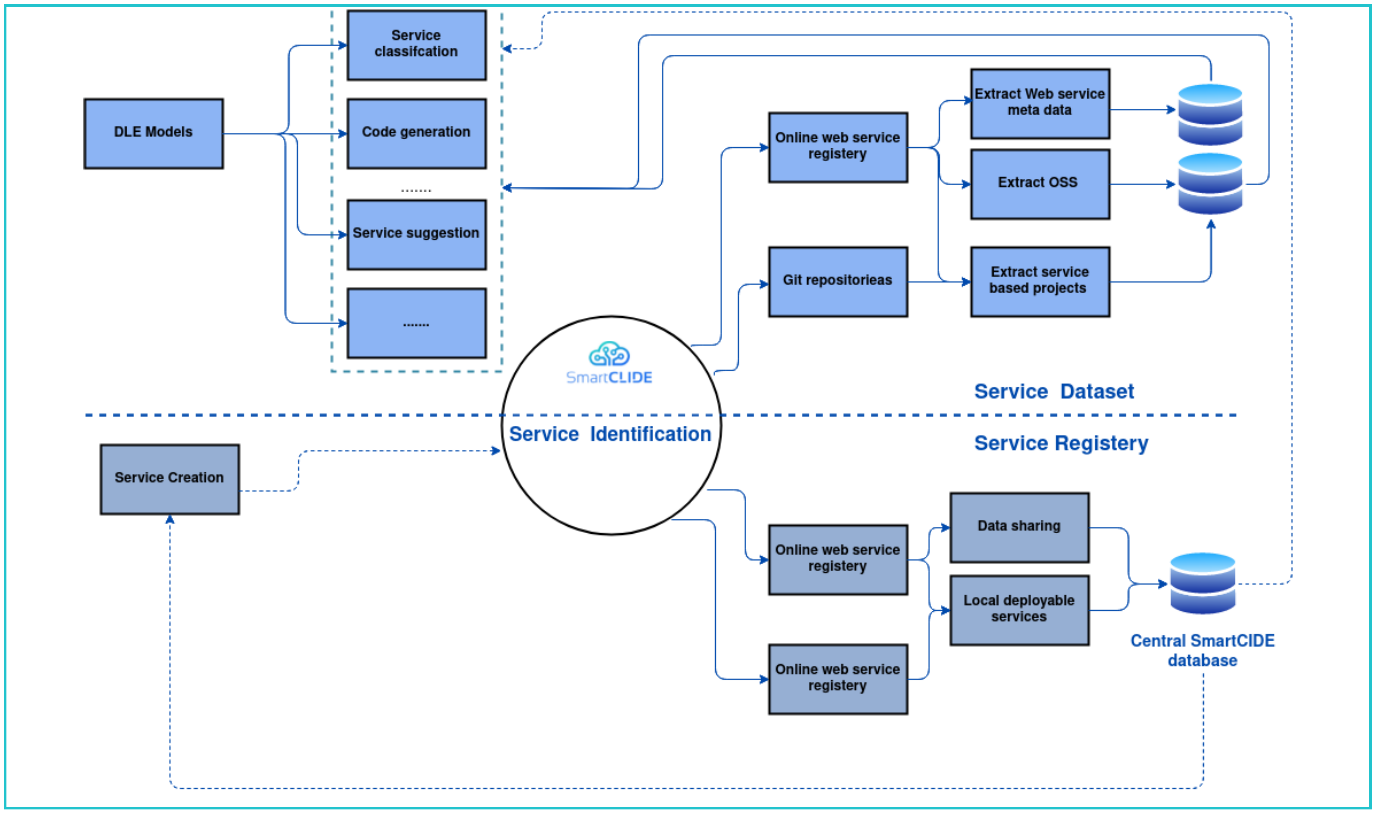


Figure 30: SmartCLIDE External Service Identification

A combination of the existing benchmark dataset and collected data enable SmartCLIDE to implement a range of intelligent learning models. The embedded learning models in SmartCLIDE seek to improve service development main tasks, which are:

1. Identifying system requirements
2. Finding and discovering service registries and providing a pool of services
3. Classifying the discovered services to identify a list of candidate services with the same functionality for particular tasks
4. Ranking selected services with the same functionality

SmartCLIDE DLE functionality has been embedded in order to improve automation in mentioned tasks. The selected AI approaches have demonstrated proper performance in software intelligence. Language modelling based on the sequence to sequence models, recommender systems, learning from existing code, and source code analysis are some instances, to name a few.

Moreover, the collected data type, service metadata, or related text directs us to mostly take advantage of text process trends, including deep learning methods such as encoder-decoder models. These models have impacted rapid developments. Therefore, SmartCLIDE has taken advantage of Transfer Learning and uses pre-trained models. The following table lists some popular deep learning models in software intelligent literature.

Table 2: Popular Related Pre-trained AI models

Pre-Trained-Models	Year	Description
OpenAI's GPT-3	2020	GPT-3 is the 3rd version release of GPT-2. This model is over 10x the size of its predecessor, GPT-2)
OpenAI's GPT-Neo	2021	Microsoft published in September 2020 that it had authorised "exclusive" use of GPT-3; others can still use the public API to receive output, but only Microsoft has control of the source code. GPT-Neo goal is to replicate a GPT-3 sized model and open source it to the public
OpenAI's GPT-2	2019	The model is that it is trained in a database of 8 million web pages. GPT2 base model has 117M parameters, GPT2-medium has 345M and the GPT2-large 774M parameters.
Bert	2019	BERT (Bidirectional Encoder Representations from Transformers) was published in 2018 and Google has announced a language model in October 2019. The significant feature of BERT is using
		BiLSTM, the most promising model for learning long-term dependencies. BERT base model has 110M parameters whereas BERT large has 340M parameters.
DistilBERT	2020	The smaller BERT version to consider resource usage and performance, has been introduced, which is smaller and runs 60% faster than BERT .

In summary, for increasing productivity with real-world data, some of the AI-based models in SmartCLIDE use pre-trained language modelings like BERT and GPT2. These models have trained on enormous data on the internet and have demonstrated acceptable results in both research and industrial communities. Yet, the individual classifiers are still considered based on available data size. The best practice for the training process is to use customised local data; nevertheless, in the beginning, we used some benchmark datasets in software intelligence academic works and available open source data. Training time, resource consumption, data storage capacity, and real-time interface interaction are other factors that DLE has to deal with to design and implement learning models.

BACKEND SERVICES

This section presents the **Backend Components and Services:**

- Source Code Repository choice,
- **Services Discovery,** Creation and Management subcomponents,
- The **Security** Assurance module and its 2 mechanisms: Vulnerability Prediction and Quantitative Security Assessment.
- The **Message Oriented Middleware** component in charge of the inter-component communication with the SmartCLIDE platform.
- The User Access Management subcomponent.
- The **Deployment** workflow and its third-party services, and the **CI/CD** infrastructure.
- Tool support for architecture pattern selection in Cloud-centric service-oriented IDEs
- Runtime monitoring and Verification
- Vulnerability prediction
- Testing Cloud-based applications

Source Code Repository

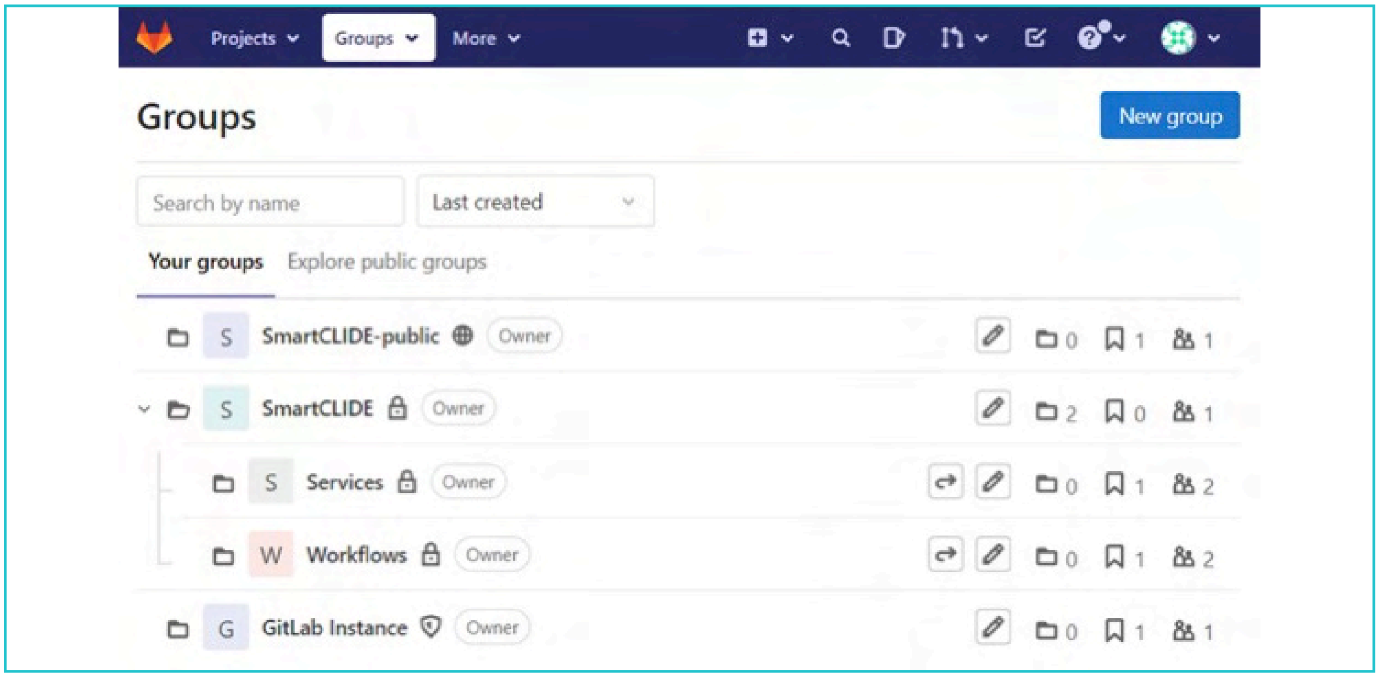
By Netcompany-Intrasoft

A consequence of the dominance of Git in the market is that the majority of development tools have excellent support for it as a version control system, either built-in or available as a plugin. Regarding the tools selected to be reused by SmartCLIDE, Eclipse Theia includes built-in git support, while workflows defined in jBPM are stored internally in Git and can be synchronized to an external Git repository.

While Git on its own provides excellent support for version control, there are several services that provide additional functionality on top, including GitHub, GitLab, and Atlassian Bitbucket. Some common additional features are:

- A web-based user interface to support git repository configuration as well as other value-added services
- Support for code review
- Support for CI/CD pipelines
- For SmartCLIDE, the consortium has chosen to use GitLab since it is available as a pre-packaged Docker image that can be deployed either on-premises or in the cloud and has a number of other features that make it useful for integration in the SmartCLIDE environment, such as:
- Integration with external security providers for access management, and Keycloak in particular, which is the chosen User Access Management platform
- RESTful and GraphQL APIs that can be used by other components of SmartCLIDE
- Native support for CI/CD
- Hierarchical organisation of Git repositories using "Groups" and "Subgroups"

As a proposed structure, the top-level SmartCLIDE group contains sub-groups named "Services" and "Workflows", each of which contains GitLab projects that contain a Git Repository plus other data for handling additional features such as merge requests and CI/CD.



Groups

SmartCLIDE-public	Owner	0	1	1
SmartCLIDE	Owner	2	0	1
Services	Owner	0	1	2
Workflows	Owner	0	1	2
GitLab Instance	Owner	0	1	1

Figure 31: Hierarchical Group Structure in GitLab

There are a number of source-code artifacts required to implement a workflow in the SmartCLIDE environment, which should all be version-controlled:

- **Workflow definitions and metadata:** Each workflow definition has its own repository on the GitLab server. Data stored in version control for the workflow includes:
- Metadata regarding the workflow (e.g., name, description, and service dependencies)
- Gherkin description of the workflow
- BPMN model of the workflow.
- **Service Source Code:** In cases where a service is written from scratch, assembled using a template, or otherwise modified from existing source code, the service should have its own Git repository on the GitLab server. It is proposed to build a library of service definitions, grouped separately from the workflow definitions, since each service has the potential to be re-used in different workflows.

If you wish to learn more about this aspect of the SmartCLIDE project, we invite you to read the public deliverable entitled "D3.1 – Early SmartCLIDE Cloud IDE Design".

Discovery of Services and Resources

The Discovery of Services and Resources backend module is responsible for collecting data on services discovered through the use of crawlers, maintaining an internal registry of services, as well as serving queries/requests for services based on service usage details and service code requests. This component communicates with the DLE (: Deep Learning Engine) to classify services and receive updates.

The Service Discovery component will have five sub-components:

Subcomponent name	Functionality
Crawlers	Collect information from web service listings, service code repositories, service registries and provide data ready to be stored in the registry.
Internal Services Index Manager	Allows to store, search and classify both discovered and new created services. This component communicates with the DLE classifier and uses the Elasticsearch API to perform searches, along with its internal SQL service registry.
Repository of discovered services	Store the records discovered by the crawler tool in .csv files while executing the retrieval requests, serving as a backup of the discovered services until they are uploaded to the internal database.
New service Creation	Allows new services, created from the Service, Composition and Testing component to be stored and classified in the service registry index.
Search Services	Using an internal SQL record and the Elasticsearch API, this component will accept search requests that it will delegate to the internal registry and then to Elasticsearch, returning the ranked services to the user based on the search.

Service Creation, Composition and Testing

The Service Creation subcomponent will be responsible for handling the creation of a new service. The component will create the required infrastructure for the development process by automatically creating and configuring functions such as version control and continuous integration. The above will be achieved by leveraging the already existing GitLab API. Apart from aiding with the creation process, the component will accompany the user through it by providing useful functionalities through interaction with other components and external tools. The user will have the ability to request functionalities by notifying Deep Learning Engine or Software Security. Furthermore, the user will be able to fetch analysis data from the Reusability Index and TD Principal and Interest subcomponents. Finally, an API will be exposed, that will allow the usage of certain functionalities when called by either the Eclipse Theia extension, JBPM Workbench or any other type of UI. The overall purpose of the component is to aid during the development process and ultimately lead to a better implemented final result.

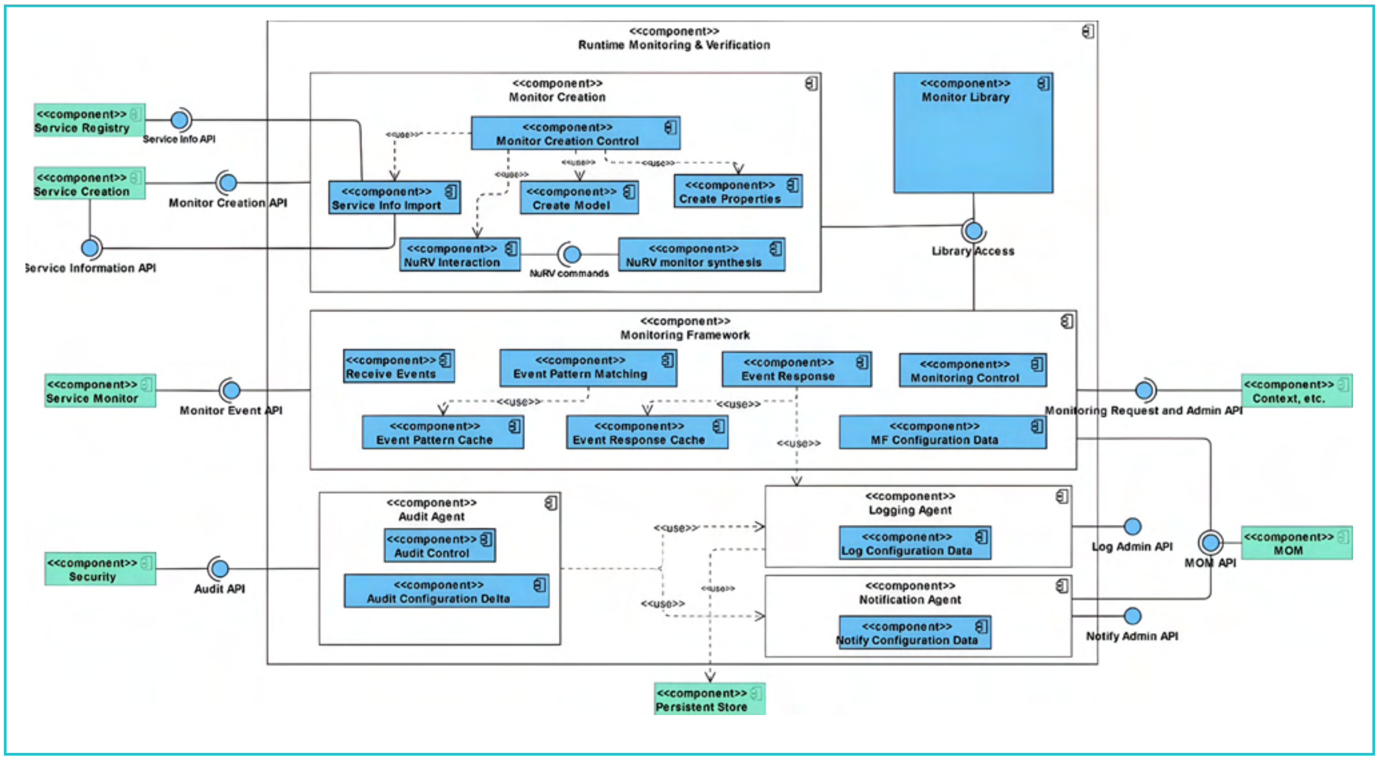
Runtime Monitoring & Verification

When services are created in SmartCLIDE the Runtime Monitoring & Verification (RMV) may be employed to generate a runtime monitor for the service. Runtime monitors supplement, at run time, other quality assurance measures such as testing and verification that are employed at development time. Particularly in SmartCLIDE, when automated methods are used to generate, or assist in the generation of, code for applications with minimal human intervention, it is possible for there to be semantic "misunderstandings" between component services composed to create the new service, or unexpected interactions of features of the components, resulting in unexpected and undesirable behaviors. Runtime verification may be able to quickly intercept misbehaving services and take a predetermined defensive or remedial action.

The monitors created for a service may be reviewed by the user and disabled or additional monitors specified and generated to be run with the service. In addition, custom monitoring services may be created as SmartCLIDE generated services, to serve other user applications or SmartCLIDE components.

The RMV has explicit support for security and context-sensitivity SmartCLIDE component in addition to synthesized monitoring for created services and the creation of bespoke monitoring services.

The RMV will incorporate and build upon several existing technologies as well as implement new features and integrate them in a novel way to support the runtime quality assurance goal of SmartCLIDE



<<component>> Runtime Monitoring & Verification
<<component>> Monitor Creation
<<component>> Monitor Creation Control
<<component>> Service Info Import
<<component>> Create Model
<<component>> Create Properties
<<component>> NuRV Interaction
<<component>> NuRV monitor synthesis
NuRV commands
<<use>>
<<component>> Monitor Library
Library Access
<<component>> Service Registry
Service Info API
<<component>> Service Creation
Monitor Creation API
Service Information API
<<component>> Monitoring Framework
<<component>> Receive Events
<<component>> Event Pattern Matching
<<component>> Event Response
<<component>> Monitoring Control
<<component>> Event Pattern Cache
<<component>> Event Response Cache
<<component>> MF Configuration Data
<<component>> Service Monitor
Monitor Event API
Monitoring Request and Admin API
<<component>> Context, etc.
<<component>> Audit Agent
<<component>> Audit Control
<<component>> Audit Configuration Delta
<<component>> Logging Agent
<<component>> Log Configuration Data
<<component>> Notification Agent
<<component>> Notify Configuration Data
<<component>> Security
Audit API
Log Admin API
MOM API
<<component>> MOM
Notify Admin API
<<component>> Persistent Store

Figure 32: Runtime Monitoring and Verification Component Diagram

Among the incorporated extant technologies are:

- The overall architectural framework of command processor, server structure, RESTful APIs, and testing from an implementation of the Next Generation Access Control standard [1] by The Open Group [2].
- The Event Processing Point (EPP) from TOG-NGAC an adaptation and extension of the EPP will form the core of the Monitoring Framework component
- The runtime verification extension [3] to the symbolic model checker [4] will form the core of the Monitor Creation component
- A recent audit API addition to TOG-NGAC will be adapted for the Audit Agent.

The RMV will include newly developed components for SmartCLIDE including:

- Create SMV model K for the service to be monitored from service specifications used by SmartCLIDE service creation
- Create property specifications in Linear Temporal Logic (LTL) from service specifications
- Logging Agent for flexible and configurable logging
- Notification Agent to provide flexible and configurable notification to SmartCLIDE components
- Monitor Library to hold various facts and rules needed by other components and modules of RMV

We provide a brief description of the function of each of the main subcomponents in the following table:

Subcomponent name	Functionality
Monitor Creation	Monitor creation utilizes service specifications of the service to be monitored to build a behavioral model and a set of essential properties.
Monitor Library	This component is a database of various facts and rules used by other modules of the RMV system. Information contained in the Monitor Library includes: – SMV (: Symbolic Model Verifier) specification patterns – LTL (: Linear Temporal Logic) property patterns
Monitoring Framework	This component is the hub and control system of RMV. It contains the Event Processing Point (EPP) which receives events and current property verdicts from the monitors running with their associated services. Received events are processed according to the Monitoring Framework Configuration Data which includes Event-Response Packages (ERP) that define event patterns and associated responses. Received events are checked against cached event patterns. When a pattern match occurs the associated response from the event response cache is executed by the Event Response Execution function.
Audit Agent	Security auditing services comprising the abilities to: – Define a set of auditable events – Select a subset of the auditable events to be collected in the audit log – Define the format of the audit log record – Generate an audit log record in response to a reported event – Store audit records in a persistent audit log file through the Logging Agent – Manage the audit service and audit log files – Generate audit alarms to be delivered through the Notification Agent

Logging Agent	Filter and store log messages in persistent log storage according to the Logging Configuration Data
Notification Agent	Send direct notifications to SmartCLIDE components according to the Notification Configuration Data

If you wish to learn more about this aspect of the SmartCLIDE project, we invite you to read the public deliverable entitled "D3.1 – Early SmartCLIDE Cloud IDE Design".

References

[1] "International Committee for Information for Information Technology Standards – Cyber security technical committee," 1. American National Standard for Information Technology Next Generation Access Control (NGAC)." ANSI, INCITS 565-2020, April 2020.

[2] "NGAC policy tool, policy server, and EPP Release note for v0.4.7 development version," Rance DeLong, The Open Group, July 2021.

[3] A. Cimatti, C. Tian, and S. Tonetta, "NuRV: A nuXmv Extension for Runtime Verification," in Runtime Verification, Cham, 2019, pp. 382–392. doi: 10.1007/978-3-030-32079-9_23.

[4] R. Cavada et al., "The nuXmv Symbolic Model Checker," in Computer Aided Verification, Cham, 2014, pp. 334–342. doi: 10.1007/978-3-319-08867-9_22

Security

By CERTH

Software security is a critical consideration for software development companies who want to provide safe and dependable software to their clients [1]. Modern software applications are typically accessible through the internet and handle sensitive data. As a result, they are continually vulnerable to harmful assaults. Exploiting a single vulnerability can have far-reaching repercussions for both the end-user (e.g., information leakage) and the organization that owns the affected software (e.g., financial losses and reputation damages) [2]. As a result, the software industry has shifted its focus towards creating proactive approaches that may give developers suggestive information about the security quality of their programs by detecting susceptible hotspots in the source code.

The Vulnerability Prediction (VP) mechanism is one such system that enables the prediction and mitigation of software vulnerabilities early in the development cycle. By assigning limited test resources to potentially risky items, VP models (VPM) can be utilized to prioritize testing and inspection efforts. Several VPMs have been developed throughout the years that use a variety of software elements as inputs, such as software metrics, static analysis warnings, and a text mining approach known as bag-of-words (BoW) [1], [3]. Although these models have shown encouraging outcomes, there is still room for improvement. Static analysis warnings contain a high number of false positives in addition to severe alarms. The BoW technique appears to produce better results than static analysis alerts and the usage of software metrics; however, it is overly reliant on the software project used for model training. As a result, current research has switched its attention to more complex approaches for detecting patterns in source code that signal the presence of a vulnerability. They concentrate on collecting information from a specific software application's raw source code or from abstract representations of its source code, such as their Abstract Syntax Tree.

Using the raw text of the source code in the form of sequences of instructions, this work creates deep-learning (DL) models capable of predicting whether a software component is susceptible or not, employing approaches from the fields of natural language processing (NLP) and text classification. We used approaches from the NLP discipline for this aim. The source code is seen as text, and the vulnerability assessment work, like sentiment analysis, is regarded as a text classification problem. So, using NLP techniques such as Bidirectional Encoder Representations from Transformers (BERT) [4], data pre-processing and transformation to sequences, and training DL models (e.g., recurrent neural networks) suitable for analyzing sequential data, we detect potentially vulnerable components using a binary classifier trained primarily on text token sequences from the source code. Furthermore, software measurements acquired by static code analyzers, in conjunction with text mining approaches, might be utilized to improve the prediction performance of the models.

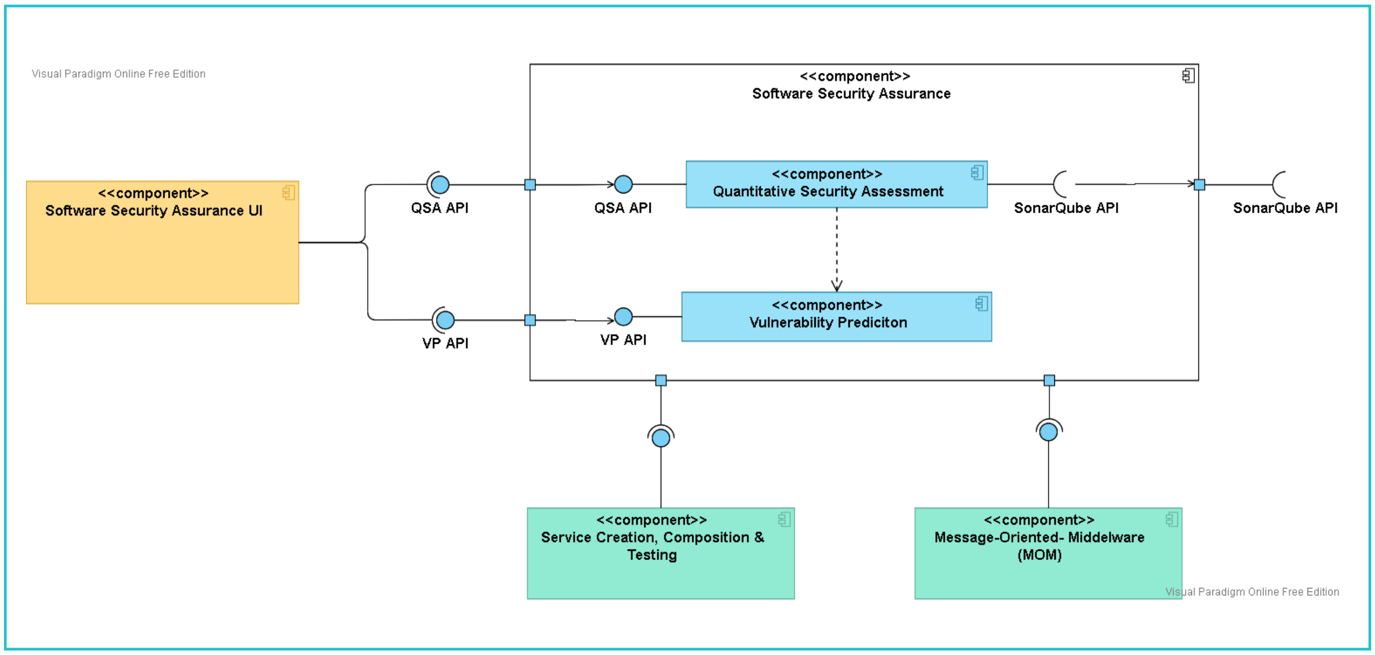

Visual Paradigm Online Free Edition
<<component>>
Software Security Assurance
<<component>>
Software Security Assurance UI
QSA API
QSA API
<<component>>
Quantitative Security Assessment
SonarQube API
SonarQube API
<<component>>
Vulnerability Prediciton
VP API
VP API
<<component>>
Service Creation, Composition & Testing
<<component>>
Message-Oriented- Middelware (MOM)
Visual Paradigm Online Free Edition

Figure 33: Software Security Assurance Module

Subcomponent name	Functionality
Quantitative Security Assessment	Responsible for assessing security level of software applications based on Security Assessment Model
Vulnerability Prediction	Is responsible for predicting security issues (i.e., vulnerabilities)

If you wish to learn more about this aspect of the SmartCLIDE project, we invite you to read the public deliverable entitled "D3.1 – Early SmartCLIDE Cloud IDE Design".

References

[1] M. Siavvas, E. Gelenbe, D. Kehagias, and D. Tzovaras, “Static Analysis-Based Approaches for Secure Software Development,” in Security in Computer and Information Sciences, Cham, 2018, pp. 142–157. doi: 10.1007/978-3-319-95189-8_13.

[2] E. Gelenbe et al., “NEMESYS: Enhanced Network Security for Seamless Service Provisioning in the Smart Mobile Ecosystem,” in Information Sciences and Systems 2013, Cham, 2013, pp. 369–378. doi: 10.1007/978-3-319-01604-7_36.

[3] S. M. Ghaffarian and H. R. Shahriari, “Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey,” ACM Comput. Surv., vol. 50, no. 4, p. 56:1-56:36, Aug. 2017, doi: 10.1145/3092566.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota, Jun. 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.

Intercommunication

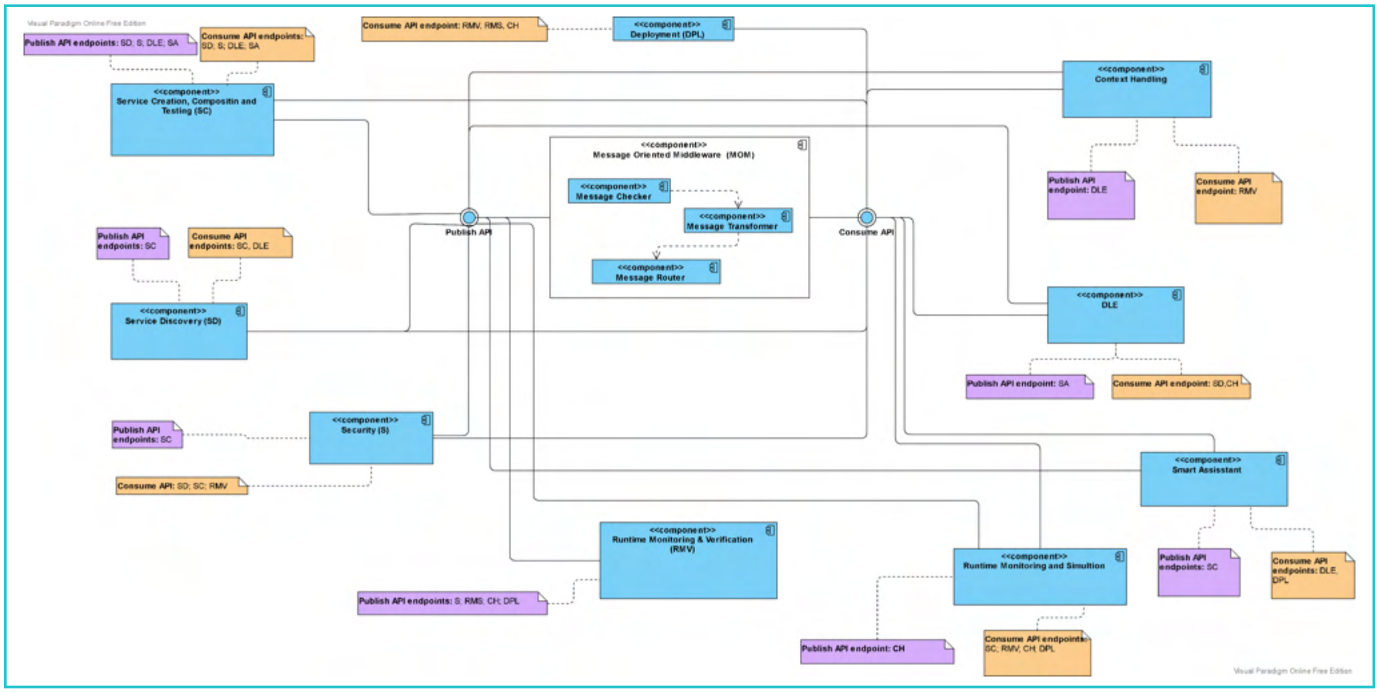
By CERTH

Message Oriented Middleware (MOM) component is responsible for inter-component communication with the SmartCLIDE platform. The MOM is designed and implemented as a message broker which is a piece of software that enables applications, services, and systems to communicate with one another to exchange information [1]. This communication is achieved by translating messages between formal messaging protocols, allowing independent services written in different languages or implemented in various platforms to interact with each other.

SmartCLIDE's MOM component offers standardized means of handling the data flow between the components of the SmartCLIDE platform. Thus, developers using the SmartCLIDE platform can focus on the core logic of the application. MOM can validate, route, store, and deliver messages to the appropriate destinations, allowing senders to issue messages without knowing where the receivers are and whether they are active or not, thus facilitating the decoupling of services and processes within systems.

There are several message brokers available, with popular choices being Apache Kafka and RabbitMQ. For the implementation of the MOM component, we have used the official RabbitMQ Docker image [2] in order to run the RabbitMQ server inside a Docker container. This is the easiest way to have a RabbitMQ instance up and running and enhances portability. For making RabbitMQ available to the other SmartCLIDE components, we have set up a RESTful API with the help of Spring Boot, thus offering HTTP access to the message broker.

MOM component diagram is presented in the next diagram. The MOM component resides in the center of the system architecture and comprises three sub-components, namely the Message Checker, the Message Transformer, and the Message Router. The Message Checker is the first point of interaction when communicating with the MOM component and verifies the validity of the incoming/outcoming messages while the actual routing of the messages is implemented by the Message Router sub-component. The Message Transformer modifies each message accordingly so that it can be parsed at both ends (publisher and consumer).



<<component>> Message Oriented Middleware (MOM)
<<component>> Message Checker
<<component>> Message Transformer
<<component>> Message Router
Publish API
Consume API
<<component>> Service Creation, Compositin and Testing (SC)
<<component>> Deployment (DPL)
<<component>> Context Handling
<<component>> Service Discovery (SD)
<<component>> DLE
<<component>> Security (S)
<<component>> Smart Assistant
<<component>> Runtime Monitoring & Verification (RMV)
<<component>> Runtime Monitoring and Simultion

Figure 34: MOM Component Diagram

Subcomponent name	Functionality
Message Checker	MoM will be able to check if the exchanged messages, either at the sender's or at the receiver's end, comply with a specific format.
Message Transformer	MoM will transform the data/messages from the sender's native format to the receiver's native format.
Message Router	MoM should support several message routing policies and message delivery guarantees (e.g., at-most-once, and exactly-once).

If you wish to learn more about this aspect of the SmartCLIDE project, we invite you to read the public deliverable entitled "D3.1 – Early SmartCLIDE Cloud IDE Design".

References

[1] "What are Message Brokers?," Aug. 11, 2021. https://www.ibm.com/cloud/learn/message-brokers (accessed Sep. 04, 2021).

[2] "Rabbitmq – Official Image | Docker Hub." https://hub.docker.com/_/rabbitmq (accessed Sep. 06, 2021).

User Access Management

By CERTH

User Access Management (UAM), also known as identity and access management (IAM), is the act of defining and managing the roles and access privileges of individual network entities (users and devices) to a variety of cloud and on-premises applications. Users include customers, partners, and employees, while devices include computers, smartphones, routers, servers, controllers, and sensors. The core objective of an IAM solution is to assign one digital identity to each individual or a device. Once that digital identity has been established, the IAM solution maintains, modifies, and monitors access levels and privileges through each user's or device's access life cycle.

In today's complex compute environments, IT departments are under increased regulatory and organizational pressure to protect access to corporate resources. As a result, they can no longer rely on manual and error-prone processes to assign and track user privileges. IAM automates these tasks and provides a seamless way to manage user identities and access all in one place. The core responsibilities of an IAM system are:

- Verification and authentication of users based on their roles and contextual information such as geography, time of day, or (trusted) networks
- Capturing and recording of user login events
- Managing and provision of visibility of the business's user identity database
- Managing the assignment and removal of users' access privileges
- Allowing system administrators to manage and restrict user access and monitor changes in user privileges

The adoption of an Identity Management system provides a wide range of benefits to organizations, such as:

- Secure access: access privileges are granted according to the selected policy, and all individuals and services are properly authenticated, authorized and audited
- Reduced risk: companies have greater control of user access, which reduces the risk of internal and external data breaches
- Ease of use: the use of an IAM framework can make it easier to enforce policies around user authentication, validation and privileges
- Reduced IT costs: businesses can operate more efficiently by decreasing the effort, time and money that would be required to manually manage access to their networks
- Meeting compliance: an effective IAM system facilitates businesses to confirm compliance with critical privacy regulations such as HIPAA, the Sarbanes-Oxley Act and GDPR

If you wish to learn more about this aspect of the SmartCLIDE project, we invite you to read the public deliverable entitled "D3.1 – Early SmartCLIDE Cloud IDE Design".

Deployment and CI/CD

By Netcompany-Intrasoft

Deployment and deployment monitoring service

This is a very first approach where we will only consider:

GitLab + “SmartCLIDE Interpreter + Jenkins + Docker + Kubernetes/AWS

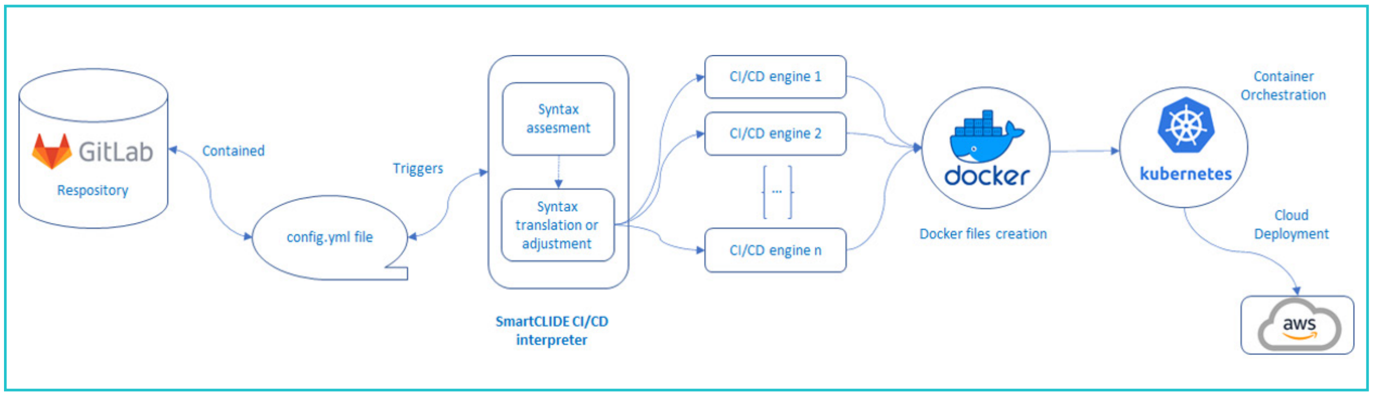


Figure 35: Set of Applications Diagram. Workflow

The deployment and deployment monitoring microservice makes use of the following elements to perform the deployment and monitoring tasks, from the GitLab-ci pipeline file until the deployment is monitored as it runs on the Kubernetes cluster, be it on any cloud infrastructure.

Element name	Role
Kairos interpreter	The Kairos interpreter is a microservice developed by our partner KAIROS whose main function is, from a GitLab-ci file, to obtain a Jenkins pipeline file.
Jenkins	Jenkins was chosen as the CD/CI engine since it is the main target of the Kairos interpreter as a CD/CI engine. In addition, since it is open-source software, it can be deployed on the developer’s machine in the development and testing tasks of the deployment microservice. As more and more organizations are using Docker to unify their build and test environments for their applications, Jenkins allows us to

	interact with Docker through default Docker support in its pipelines. On the other hand, Jenkins pipelines allow images to be built from the Dockerfile found in the main folder of the software project.
Docker registry	Docker Registry is an application that manages to store and deliver Docker container images. Registries centralize container images and reduce build times for developers. Docker images guarantee the same runtime environment through virtualization, but building an image can involve a significant time investment. Docker registry will be used as a central repository of images once they are built. It will be from this service from where the Kubernetes deployment will obtain the image of the containers to be deployed in the cluster.
Kubernetes cluster	Because Kubernetes is an open-source project, you can use it to run containerized applications in any environment without having to change your operational tools. A large community of volunteers maintains and improves Kubernetes software. In addition, many other vendors and open-source projects create and maintain Kubernetes-compatible software that you can use to enhance and extend your application infrastructure. The scope of the service also includes the use case of obtaining information about the status of the service while it is running in Kubernetes. Some of this data can be RAM memory in use, network information, or CPU usage. It is assumed that a Kubernetes cluster is running on any of the clouds, such as AWS, Azure, or Google Cloud Platform.

SmartCLIDE CI/CD

The proposed basis for the SmartCLIDE CI/CD infrastructure is the built-in CI/CD capability provided natively by the chosen version control system, GitLab. For the Continuous Integration (CI) component of this, there are several areas to consider:

- What elements of the system are subject to CI
- How CI integrates with the development strategy

When considered at the level of individual services, CI is a requirement for those services which are defined within the SmartCLIDE source code repository, i.e., services which are developed from scratch, modified from a template, or generated as source code with SmartCLIDE tooling. CI on GitLab is generally configured by means of a configuration file, namely the GitLab-ci.yml file, at the root of the corresponding source repository. Within this configuration file, the various stages of the build pipeline are defined. A build pipeline might typically involve the following stages:

- Build – compile the code in the repository
- Unit test – run unit tests
- Package – package the service into a deployable unit
- Integration test – run integration tests
- Deploy – deploy to an environment

For Continuous Delivery (CD) at the workflow definition level, the flexibility afforded by the GitLab CD functionality, with built-in support for Docker and Kubernetes deployments and the ability to run arbitrary scripts, may serve as the basis for the deployment of the composed service.

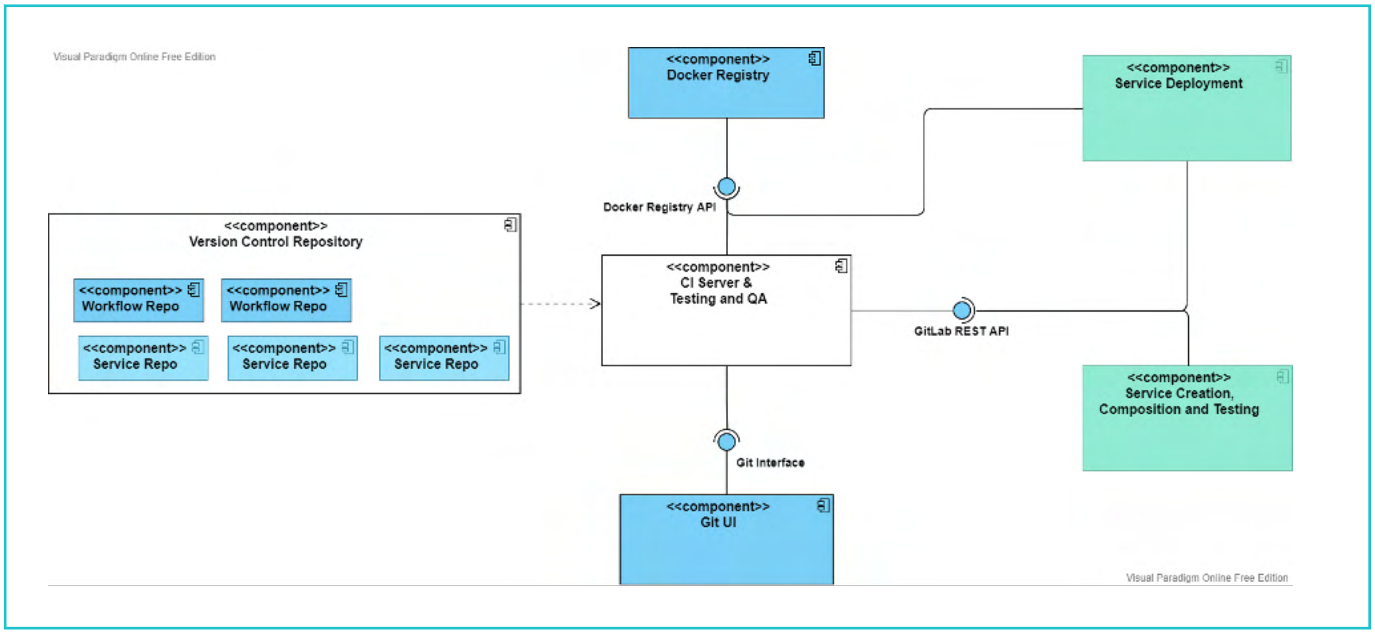

Visual Paradigm Online Free Edition
<<component>> Docker Registry
<<component>> Service Deployment
Docker Registry API
<<component>> Version Control Repository
<<component>> Workflow Repo
<<component>> Workflow Repo
<<component>> Service Repo
<<component>> Service Repo
<<component>> Service Repo
<<component>> CI Server & Testing and QA
GitLab REST API
<<component>> Service Creation, Composition and Testing
Git Interface
<<component>> Git UI
Visual Paradigm Online Free Edition

Figure 36: CI Server & Testing and QA Component Diagram

Subcomponent name	Functionality
CI Server & testing Component	Perform automated build, test, and packaging of services from source code.

If you wish to learn more about this aspect of the SmartCLIDE project, we invite you to read the public deliverable entitled "D3.1 – Early SmartCLIDE Cloud IDE Design".

Tool Support for Architectural Pattern Selection in Cloud-centric Service-oriented IDEs

By ATB Bremen

Software architecture is a central part of software engineering and plays a crucial role in the success of software applications, both in terms of business and engineering aspects. Deciding on a specific software architecture requires careful analysis of the software application's requirements and is not trivial.

Architectural patterns are general design structures that have been used successfully in software architecture design. They provide rules and guidelines to describe high-level software components and the interrelation between them, and address commonly occurring issues in software architecture design, such as limitations in software performance, availability or minimization of business risk. Architectural patterns are similar to software design patterns but have a broader scope.

Selecting an architectural pattern is a challenging task for a software architect.

Software architecture design is typically made in the early stages of a software development life cycle and is very crucial for the quality, success and further management of the software. Selecting an architectural pattern is a challenging task for a software architect. It requires not only technical knowledge about these patterns, but also expertise in deciding which pattern is the most suitable architecture for a software system (considering its requirements).

While modern software development practices benefit from strong tool support offered by IDEs (features like build automation, debugging, refactoring, code search, continuous integration and continuous deployment), the need for a support system for architectural pattern selection is still not sufficiently met in practice. In particular, software engineers could greatly benefit from tool support that assists them in their architectural pattern decision process.

To address this issue we developed a framework for architectural pattern selection (APS) that can be integrated as a tool support feature in IDEs. Our framework currently supports the following six common architectural patterns:

1. Layered architectural pattern
2. Event-driven architectural pattern
3. Microkernel architectural pattern
4. Microservices architectural pattern
5. Service-oriented architectural pattern
6. Space-based architectural pattern

In order to provide tool support for the decision-making process of architectural pattern selection, certain information about the software design and requirements must be acquired. Our framework for architectural pattern selection uses four specific categories of high-level information about the application to be developed and the architecture for it. These categories are as follows:

- **Application domain:** The general domain of the software application such as web-based systems, distributed systems, cloud computing applications, mobile applications etc.
- **Application properties:** High-level properties of the software application such as specifications of application components or business constraints.
- **Non-functional requirements:** General operational specifications of the software application such as maintainability, performance, portability, reliability and security.
- **Architectural features:** High-level properties of the software architecture such as specifications of architecture component communication, component interoperability and data volume.

These categories are explored through a survey where at least one question per category can be asked to a software developer/architect. This survey is designed to collect both the background information about the application and the preferences in terms of the operational capabilities of the application as well as the desired features of the architecture.

A scoring system associates every survey item to each of the six supported architectural patterns, to provide an evaluation. These values indicate how strongly an architectural pattern is suitable to the given specification. This evaluation is based on a comprehensive state-of-the-art analysis of architectural patterns and their relation to application domains and architecture requirements. The scoring values are used to calculate the total value of each architectural pattern. The pattern with the highest total score indicates the most suitable pattern based on the data input. The top three highest ranking patterns and their ranking can be suggested for the user's consideration.

The APS framework is implemented as a backend service that provides a REST API for the survey content and its evaluation. The API implementation is independent of the survey content and the specific scoring values used for the evaluation. The survey content is prepared and stored as a JSON object that can be retrieved and used to create and present it in an IDE. The scoring values used for the evaluation are also stored as a JSON object. Both of these JSON objects are configurable within the API if desired. This makes the backend API generic in terms of API functionality and it can be integrated in IDEs to offer tool support for architectural pattern selection.

The APS framework and its backend API are currently being integrated within the SmartCLIDE research project. Architectural pattern selection is supported in the SmartCLIDE IDE as part of the service creation process using the user frontend of the IDE. The service creation flow begins with the user starting to create a new service, which is followed by the selection of a Git system to be used together with corresponding credentials. After the user provides the details of the service to be created the next step is architectural pattern selection. This step provides assistance to the user if the user decides to choose an architectural pattern for their service. This assistance is optional and is provided via the APS wizard.

The APS wizard first provides a list of supported architectural patterns to the user to choose from. The user can choose one pattern from this list and proceed to the next step. In case the user does not know which architectural pattern to choose, they can further use the APS wizard to receive a list of suggestions which patterns would be most suitable for their service based on high-level information about their service that they can provide at this stage. If the user decides to skip the selection or the application of an architectural pattern, the user front end finishes the service creation.

The backend API will be extended with the implementation of the architectural pattern application that finalizes the service creation flow in the IDE. The APS framework will be evaluated in various industrial use cases of the SmartCLIDE project. Based on the use case results, the survey content and evaluation values can be improved and re-configured in the IDE if necessary. Further future work includes increasing the number of supported architectural patterns. Currently, all supported architectural patterns are individual patterns. It would be desirable to support combinations of individual patterns as well.

Runtime Monitoring and Verification (RMV)

By OpenGroup

SmartCLIDE offers services to accelerate the creation and deployment of Cloud solutions by providing the ability for non-programmers to construct applications and new services using smart automation. One of the backend services provided by SmartCLIDE is runtime monitoring and verification (RMV) which in conjunction with automated testing is applied to assure the quality of the created services. In this paper we describe the objectives of RMV, and provide an overview of the approach and the benefits.

SmartCLIDE Quality Assurance

SmartCLIDE constructs new services according to the user's specifications from pre-existing and bespoke components. Supplementing the construction of new services, SmartCLIDE's strategy for quality assurance (QA) of user-constructed services includes both development-time and runtime quality assurance for functional and non-functional properties. In addition to the expected functional behavior of a service, key characteristics such as security, safety, privacy, resilience and reliability are general categories of runtime quality attributes that may be required of the service. Runtime QA is applied along with design-time QA, development testing, verification and qualification testing to assure that the needed quality attributes have been achieved.

Assurance of the correctness of a service may be addressed largely by the manner of its construction, giving rise to the term correct by construction. To achieve it a rigorous methodology is required, typically supported by automation[3] and tools. By automating the construction process certain sources of potential flaws may be systematically eliminated. In SmartCLIDE automation extends to AI-powered assistance in the selection and composition of components. SmartCLIDE can already make some claims to correctness by construction because the automation is systematic. However, the details of the construction methods may not be rigorous enough to extend correctness by construction to every functional or non-functional claim that could be made for a SmartCLIDE-constructed service.

Whilst, runtime QA is also a concern for entirely human-fashioned software artifacts, it may be even more beneficial for software that is constructed without human involvement and scrutiny of every design and implementation decision. By reducing the development effort through automation some of the detailed expert human scrutiny that the service development would otherwise receive will likely not occur. Subtle semantic anomalies and "corner cases" may go undetected when automation uses service specifications to construct service implementations from diversely sourced and specified components, and only surface when run-time execution behaviors are observed.

Complementary Quality Assurance Methods

Among the methods that that could have been utilized for quality assurance of SmartCLIDE created services are: correctness by construction, formal verification (FV), automated testing, and runtime verification RV. All of these methods, except for formal verification, are employed in SmartCLIDE. FV provides construction-time assurance that increases confidence that runtime behavior will not include unwanted effects by exploring all possible executions a priori.

The application of FV, even the "fully-automated" kind, is typically expensive: being laborintensive and requiring specialized expertise. It must be re-performed whenever the model is changed. Furthermore, it typically only verifies the model (an abstraction) of the implementation as opposed to the actual executable implementation. Due to these considerations we do not further consider FV as a viable routine activity in SmartCLIDE.

Conventional testing is one of the standard methods of discovering and correcting the sources of errant behaviours, and this method is also applied in SmartCLIDE by doing automated testing for SmartCLIDE-created services in addition to the unit and integration tests of the SmartCLIDE components themselves. Testing of SmartCLIDE-created services has the benefit that the actual service implementation is exercised in the tests rather than a model of the implementation as would be the case in FV. Testing involves identification of a finite number of test scenarios and test cases. As always, the issue with finite testing of a reasonably complex system, which has a potentially, and practically infinite number of distinct behaviours, is one of confidence in the adequacy of the testing, in particular that of chosen test cases and the test data. When test cases and test data are chosen automatically an additional source of automation-induced error or incompleteness is a source of adequacy concern.

Another potential source of runtime misbehavior has nothing to do with the construction or the functionality of a service but with the assumptions that underlie, possibly implicitly, the implementation of a component or a service. The implementation is only valid as long as these assumptions are satisfied and maintained. When assumptions are violated, either through incorrect composition of components, or through dynamically changing conditions at runtime, the implementation is likely to misbehave or completely fail.

Runtime monitoring and verification (RMV) is an aspect of the SmartCLIDE QA strategy that is used primarily at run time but also may be beneficial in the latter stages of development.

RMV is able to check the monitored service at every step to confirm that it's behavior in the current run is consistent with its specifications and the, necessarily limited, results of prior finite testing.

[3] Here "automation" or "automated" are used for processes that are fully or partially automated.

One of the main strengths of runtime verification is that it has the potential to detect a deviation from the required behaviour due either to an incorrect implementation of the specified behavior or to the, possibly dynamic, invalidity of an assumption.

Assurance of the runtime behaviour is addressed by validating that the service actually exhibits behavior consistent with the user's specification and with development-time test results, and that the assumptions made about the runtime environment, which were made at design time and thus built into the construction of the service, continue to be valid as the service executes.

Runtime Monitoring & Verification

Figure 37 shows an overview of the components of the SmartCLIDE RMV subsystem. The RMV subsystem interacts with other SmartCLIDE components through Message Oriented Middleware (MoM) or direct IPC, and uses an external tool, NuRV [CTT19b], to generate property monitor state machines. Property monitors are synthesized from a formal model of the nominal behavior of the created service and a specification of required properties using the method of assumption based runtime verification (ABRV) [CTT19a].

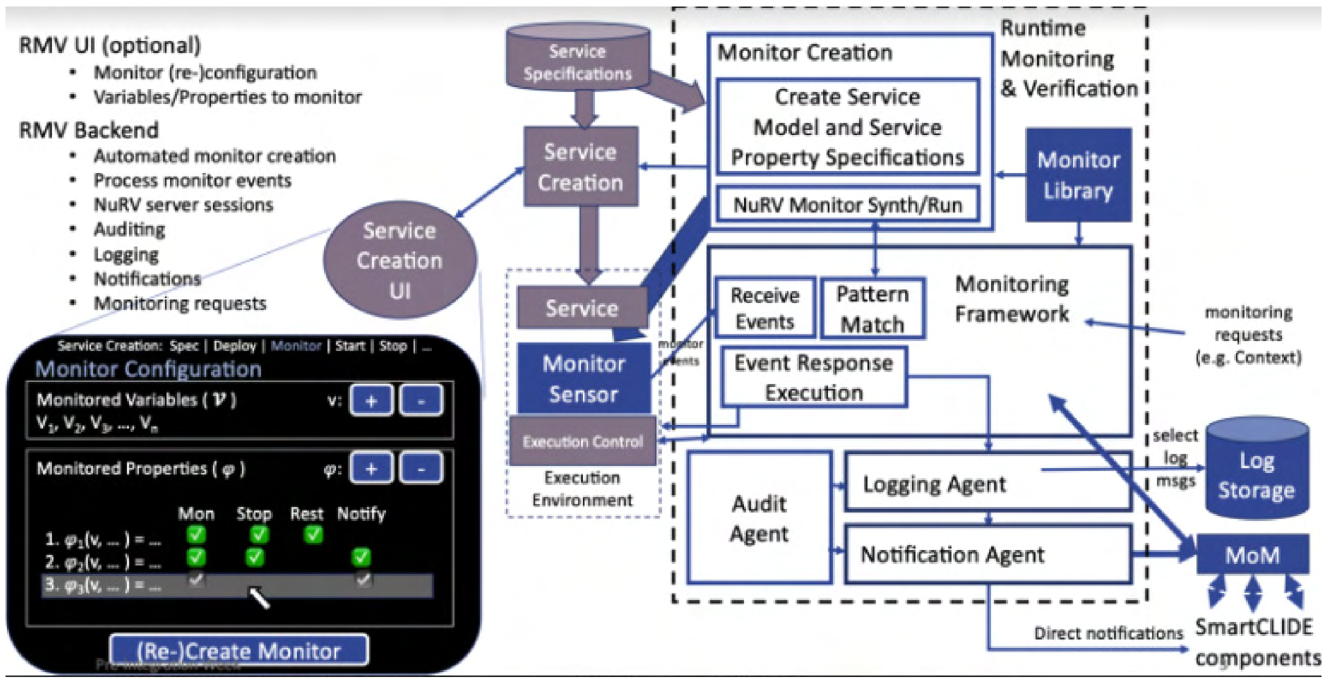


Figure 37: RMV Subsystem Overview

These components include:

1. Monitor Creation - Uses a Service Specification provided from SmartCLIDE along with elements contained in the Monitor Library to construct a property monitor using the NuRV monitor synthesis tool, and a configuration vector for the Monitor Sensor. It stores information about the created monitor in the Monitor Library.

2. Monitor Sensor - A component with versions implemented in various programming

languages that provides presence for the monitor within the SmartCLIDE-created service. When the service starts the Monitor Sensor is customized with specifics from the configuration vector. Subsequently the Monitor Sensor generates messages to Monitor Event Processing conveying information about the configured variables it shares with the monitored service.

3. Monitor Event Processing - Receives messages from the Monitor Sensor, which it processes according to the configuration for that monitor that is stored in the Monitor Library. The configuration may indicate that the values of logical conditions, based on the values of variables within the monitored service, are to be sent to a NuRV property monitor that will return a verdict on whether the monitored property is satisfied, violated, or (as yet) unknown. The result may be sent to other SmartCLIDE components that have registered for notifications.

4. Auditing, Logging, and Notification - Provides the ability to distribute monitoring data and results, to record security-relevant (or other property related) events in a persistent log, and to provide a consolidated auditing, logging and notification service to registered SmartCLIDE or application components.

5. Monitor Library - Contains global definitions and patterns for monitor construction as well as information about the specific monitors that have been constructed. The monitor library is access both at monitor construction time and at monitor execution time.

6. RMV User Interface within the SmartCLIDE Service Creation UI - An optional user interface that can be used by a service developer to modify the configuration of a service monitor, to enable/disable monitoring actions, add/delete monitored variables and properties, and regenerate a modified monitor. Without the UI such changes can also be achieved by editing the generated monitor's configuration vector.

In addition to the ability to monitor created services to assure that they operate within their specifications, the RMV framework provides the capability to construct bespoke monitoring services using the RMV monitor sensor to gather runtime data, that may be used in arbitrary ways by other system services or as part of application services.

References

[CTT19a] Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-based runtime verification with partial observability and resets. In Bernd Finkbeiner and Leonardo Mariani, editors, Runtime Veri cation, pages 165{184, Cham, 2019. Springer International Publishing.

[CTT19b] Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Nurv: A nuxmv extension for runtime verification. Berlin, Heidelberg, 2019. Springer-Verlag.

Vulnerability prediction based on Text Mining and BERT

By CERTH

Vulnerability Prediction – Importance and Challenges

Building secure software is highly important for both the end users and the owning enterprises. Nowadays, software controls critical daily activities, and therefore a security breach could lead to important implications both to user security (or even safety), and to a company's reputation and finances. To this end, software development companies have shifted their focus towards the security-by-design paradigm in order to build software that is highly secure from the ground up. In order to achieve this, several tools are employed during the development process, which enables detection and elimination of potential vulnerabilities.

Vulnerability prediction is responsible for the identification of security hotspots, i.e., software components that are likely to contain critical vulnerabilities.

One important mechanism that facilitates the identification of vulnerabilities in software is vulnerability prediction. Vulnerability prediction is responsible for the identification of security hotspots, i.e., software components that are likely to contain critical vulnerabilities. This is achieved through the construction of vulnerability prediction models (VPMs), which are mainly machine learning models that are built based on software attributes retrieved primarily from the source code of the analysed software (e.g., software metrics, text features, etc.). The results of the vulnerability prediction models are highly useful for developers and project managers, as they allow them to better prioritise their testing and fortification efforts by allocating limited test resources to high-risk (i.e., potentially vulnerable) areas.

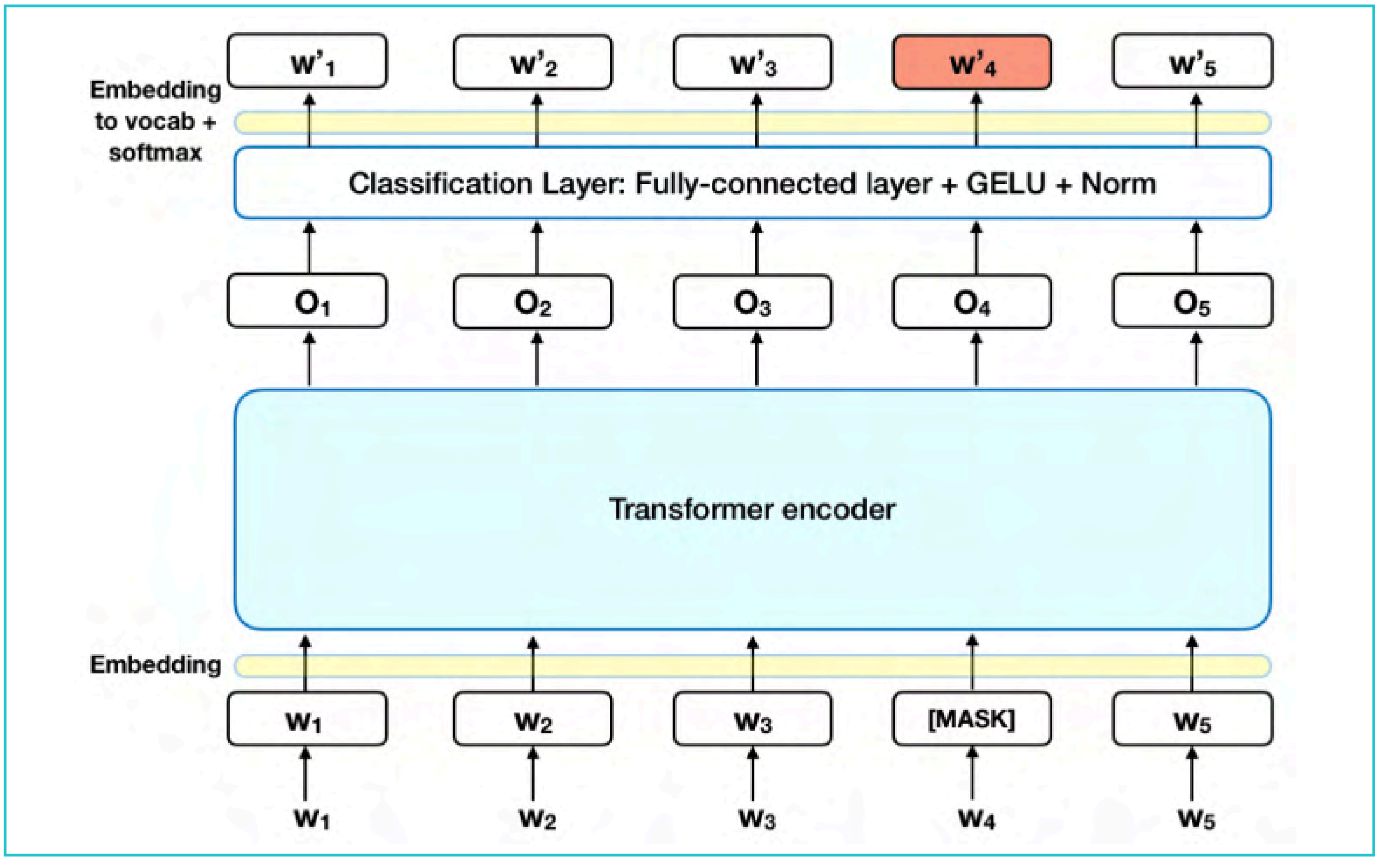
Among the existing solutions, text mining-based VPMs have demonstrated the best predictive performance. The majority of the text mining-based models that have been proposed in the literature so far are based on the concept of Bag of Words (BoW), which is actually a vector with the tokens (i.e., keywords) that are found in the source code along with the number of their occurrences, as well as on the concept of word token sequences (utilising also word embedding techniques for their representation), which corresponds to the sequences of the instructions in the analysed source code. Despite their promising results, these solutions have not demonstrated perfect predictive performance, which could allow them to be used reliably in practice, and therefore there is room for improvement. Recently, more advanced concepts have started being investigated in the literature in order to further enhance the predictive performance of text mining-based VPMs. One interesting direction which has recently started gaining the attention of the research community []-[], is the examination of whether the adoption of transformers, such as the Bidirectional Encoder Representations from Transformers (BERT) and its alternatives, could lead to more accurate vulnerability prediction.

To this end, we developed deep-learning (DL) models capable of predicting whether a software component is vulnerable, using the raw text of the source code in the form of sequences of instructions, utilising methods from the field of natural language processing (NLP) and text classification. In other words, we focused on building text mining-based VPMs utilising the popular concept of word token sequences and deep learning. We also examined whether the adoption of BERT could lead to sufficient vulnerability prediction models.

What is BERT?

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google. BERT makes use of Transformer. In its vanilla form, Transformer consists of two separate mechanisms: an encoder that reads the text input and a decoder that generates a prediction for the task. Because the goal of BERT is to generate a language model, only the encoder mechanism is required. The Transformer encoder reads the entire sequence of words at once, as opposed to directional models, which read the text input sequentially (left-to-right or right-to-left). As a result, it is regarded as bidirectional, though it would be more accurate to describe it as non-directional. This feature enables the model to learn the context of a word based on its surroundings (left and right of the word).

The Transformer encoder is described in detail in the figure below:



W'1
W'2
W'3
W'4
W'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
W1
W2
W3
[MASK]
W5
W1
W2
W3
W4
W5

Figure 38: Transformer encoder

As can be seen by the Figure above, the input of BERT is a series of tokens that are embedded into vectors before being processed by the neural network. The output is a sequence of H-dimensional vectors, each vector corresponding to an input token with the same index. The vectors that are produced by BERT can be utilized for building machine learning models for any classification problem, including vulnerability prediction, as we investigate in the present work.

Vulnerability Prediction Models using Text Mining and BERT

For the purposes of the present work, we utilised two popular vulnerability datasets proposed by the National Institute of Standards and Technology (NIST) and the OWASP, which contain examples of vulnerable and clean software components written in Java and C++ programming languages. For the case of C/C++ we utilised the Juliet dataset proposed by NIST, which contains 7651 source code files, 3438 of which are considered as vulnerable and the rest 4213 are considered as clean. For the case of Java, we utilised the OWASP Benchmark, which contains 1415 vulnerable class files and 1325 class files considered as clean.

For each dataset, the source code files were initially cleansed (i.e., comments were removed, literals were replaced with generic values, etc.) and subsequently tokenized in order to retrieve the sequences of their tokens. In order for these vectors to be used for building vulnerability prediction models, they need to be turned into numerical values, since the majority of the machine learning algorithms, including neural networks that are our main focus, operate on numerical inputs. More specifically, integer encoding was employed in order to turn the tokens into integers, and then the embedding vectors were produced. The embedding vectors are, in fact, the numerical representation of the text tokens, which can be used as inputs for our models.

In order to construct VPMs based on the selected datasets, we have used a pre-trained BERT model. Actually, it is the BERT for sequence classification pre-trained model. It belongs to the category of BERT base models with respect to their size. The model parameters, both those of the pre-trained model and those derived after fine-tuning it for the case of vulnerability prediction that we investigate in the present analysis are shown below:

Parameters	Values
Number of layers	12
Hidden size	768
Total parameters	110M
Learning rate	2e-5
Number of epochs	2-4
Batch size	2

The VPMs that were implemented both for the Java dataset and also for the C++ dataset were then evaluated with respect to their predictive performance. For the evaluation of the models, we employed the 10-fold cross-validation technique. As a measure of predictive performance, we decided to use the F2-score. The reasoning behind the selection of this evaluation metric is that the F2-score takes into account both the Recall and the Precision of the produced model, but puts more emphasis on the Recall, which is more important for vulnerability prediction since it is important for a VPM not to miss existing vulnerabilities. The results are summarized in the table below:

Model	F2 score (%)
Java	70.01
Cpp	78.73

The results of the experiments indicate that the models can identify vulnerabilities in the software to a satisfying degree. More specifically, the F2-score in both cases was found to be above 70%, which is considered sufficient in the literature, and for the case of C++ the F2-score is close to 80%, which is considered high. This suggests that the utilisation of BERT may lead to VPMs with sufficient predictive performance. In the rest of the project, we will further examine the capacity of BERT to be used in vulnerability prediction. More specifically, (i) additional datasets will be considered in order to investigate the generalizability of these observations, (ii) BERT alternatives like codeBERT will be also examined in order to see if more code-related models lead to better results, and (iii) a comparison between models utilising BERT and models based simply on text mining approaches (e.g., BoW and token sequences) without dedicated transformations will be conducted.

Testing Cloud-Based Applications

By Kairos DS

Cloud-based solutions are all the rage these days. The cloud approach is becoming extremely popular in many business areas due to advantages such as scalability, enhanced productivity, better traffic and transaction management, and significantly lower equipment costs. Moreover, a cloud-based solution makes digital operations more streamlined and provides businesses of any size with greater flexibility.

This migration of applications to the cloud has made software testing become an essential part of the business cycle. The switch to distributed and component-based applications, which is the basis for the touted flexibility and scalability, has also introduced additional layers of complexity and potential points of failure and communication, making testing cloud-based systems a vital business function.

Types of Tests

Generally speaking, every software application development must involve several types of testing distributed along the lifecycle of the product. The purpose of all such testing is to ensure the product meets both functional and non-functional requirements and to deliver a high-quality end product that will delight users. Typically, the types of testing that any application should go through are the following:

Functional testing

Functional testing ensures that the product actually provides all the services and functionalities as advertised and that the business requirements are being met. The main types of functional testing are:

- Component and Unit Testing: This kind of test is performed by developers to validate specific functionality for each unit of an application. During unit testing, each unit of code and component is tested in isolation to make sure that it works as intended and provides the expected results.
- Integration Testing: This ensures that the modules of an application are working fine and helps verify the combined functionality. Integration tests allow operational commands and data to act as a whole system, rather than as individual components. This type of testing is especially relevant to UI operations, operation timing, API calls, data formats, and database access.
- Acceptance Testing: These tests are performed by a selected group of end-users that will be given access to a functional version of the application and will validate whether the application is good enough (accepted) or not. In other words, they indicate if the application meets the business objectives.

However, for cloud-based products, it's essential to make sure that the product (or service) not only meets its functional requirements but also the non-functional ones. So a strong emphasis needs to be laid on non-functional testing as well.

Non-functional testing

Non-functional testing focuses on verifying cloud computing characteristics and features:

- **Security Testing:** A cloud offering must guarantee that the data and resources of the system are protected from any unauthorized access, but it also must be protected from threats or misuses that can take the entire system down. This can be complex and, at a minimum, involves the following:
    - ◊ Vulnerability scanning: This is done through automated software to scan a system against known vulnerability signatures.
    - ◊ Security scanning: Identifies network and system weaknesses, and provides a basis for defining solutions for reducing these risks. Both manual and automated scanning can be performed.
    - ◊ Penetration testing: This kind of testing simulates an attack from a malicious hacker. It involves the analysis of a particular system to check for potential vulnerabilities to an external hacking attempt.
    - ◊ Risk assessment: This is an assessment of the security risks that is made at a broad organizational level, involving analysis of the software itself, as well as the processes and the technologies used. Risks are classified as Low, Medium, and High. The result of this testing is a list of recommended controls and measures to reduce the risks.
    - ◊ Ethical hacking: This involves hacking into the software systems to understand vulnerabilities. Unlike malicious hackers, who steal for their own gain, the underlying intention is to expose security flaws.
- **Multi-tenancy Testing:** Multi-tenancy refers to a cloud-based service that accommodates multiple clients or organizations. The service is typically customized for each client and provides data and configuration level security to avoid any access-related issues. A cloud-based offering should be thoroughly validated for each client whenever multiple clients are to be active at a given time.
- **Performance Testing:** Performance testing checks the speed, response time, reliability, resource usage, and scalability of a cloud offering under an expected workload. A cloud-based offering should be "elastic", allowing for the increase or decrease of on-demand resource usage, while maintaining a desired throughput level. The goal of performance testing is to eliminate performance bottlenecks in the software. There are many types of performance tests. These are some of the most common:
    - ◊ Smoke tests verify that the system can handle a minimal load without problems.
    - ◊ Load tests are primarily concerned with assessing the performance of the system

in terms of concurrent users or requests per second.

◊ Stress tests and spike tests assess the limits of your system and stability under extreme conditions.

◊ Soak tests evaluate the reliability and performance of a system over an extended period of time.

- **Availability Testing:** Availability testing provides a measure of how often any given software is actually on hand and accessible for use. Cloud offerings must be available at all times. It is the responsibility of the cloud service provider to ensure that there are no abrupt downtimes. This kind of testing is primarily based on observation of the system being used along with the Quality of Service (QoS) level guaranteed by the service provider.
- **Disaster Recovery Testing:** This is a measure of the time it takes for a cloud application to recover from a disastrous failure. It may encompass certain hard measures like rolling back databases or deployments. In the case of a failure, recovery time must be low. Verification must be done to ensure the service is back online with minimal adverse effects on the client's business.
- **Interoperability Testing:** Any cloud application must work in multiple environments and platforms. It should also have the capability to be executed across many cloud platforms. It should be easy to move cloud applications and platforms from one infrastructure (as a service) to another infrastructure.

Testing Throughout the Software Development Life Cycle

Now, it becomes obvious that not all of those tests can be carried out at the same time, nor by the same set of persons, nor at the same stage of the project. However, when and how we apply all of these testing techniques plays a critical role in the quality of the resulting product.

If we look at the typical Software Development Life Cycle (the process of building software while ensuring the quality and accuracy of the software being built), it defines a series of stages and procedures. Each stage leads to the next step and produces results that move the development towards a completed product. Stages are typically defined as follows:

- **Planning stage** (also called the feasibility stage) is the phase in which developers plan for the upcoming project. Here, the problem to be solved and project scope are defined, along with determining project objectives.
- **Requirements analysis stage,** includes gathering all the specific details required for a new system, as well as determining initial prototype ideas.
- **Design and prototyping stage,** where developers will outline high level application requirements, along with more specific aspects, such as:

◊ User interfaces

◊ System interfaces

◊ Network and network requirements

    - ◊ Databases

- **Development stage,** where developers actually write code and build the application according to the earlier design documents and outlined specifications.
- **Testing stage,** where software is tested to make sure that there aren't any bugs, and that the end-user experience will not be negatively affected at any point. During the testing stage, developers will go over their software with a fine-tooth comb, noting any bugs or defects that need to be tracked, fixed, and later retested.
- **Integration and implementation (or deployment) stage,** where the system will be integrated into its environment and eventually deployed. After passing this stage, the software is theoretically ready for market and may be provided to any end-users.
- **Operations and maintenance stage,** where developers are responsible for implementing any changes that the software might need after deployment, as well as handling issues reported by end-users.

But how do the different kinds of tests relate to these stages? The following table attempts to represent the existing relationship between the SDLC stages and each type of testing.

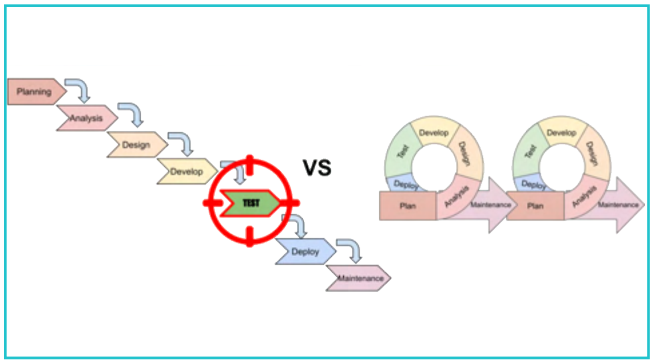
Phase	Kind of tests
Planning Stage	–
Requirements Analysis Stage	–
Design and Prototyping Stage	Process testing
Software Development Stage	Unit testing Component testing
Software Testing Stage	Acceptance testing Exploratory testing Regression testing Security testing
Implementation/Integration	Integration testing Smoke testing
Operations and Maintenance Stage	Performance testing Compatibility testing Recovery testing Availability testing

Looking at the stages, one could think that all tests will happen on the so-called “testing stage”. This is mostly true in a typical waterfall model of software project management, where a phase-only begins when the previous one has already finished. The fact that tests are left to the later stages when development is –theoretically- finished, is usually the main cause of project failures following this methodology, due to the difficulty (or even the impossibility) of applying proper corrections at such a late stage.

Fortunately, modern software development methodologies, especially the agile ones, attempt to fix this with two breaking changes:

- short feedback loops, i.e. show working things early and repeatedly, so that any
- misunderstanding or deviation can be tackled soon.
- test from the beginning, so that any defect can be found and fixed as soon as possible.

In short, agile methodologies look for integrating all — or as many as possible — of the activities inside the development iterations, because everything done within one iteration provides feedback for the next iteration.



In this light, we can see that all functional testing is actually carried out by developers during the development phase. Following the modern methodologies, developers leverage techniques like TDD (Test-Driven Development) for crafting unit tests, component tests, and integration tests, which also help them get good internal quality. They also write regression tests to ensure that any previously fixed bug does not come to life again. Acceptance tests for each task are also included to ensure that the functionality works and is properly implemented.

All of these types of tests are not only fully covered and automated, they are perfectly assumed by the developers and completely integrated into their regular day-to-day work. This means that the development team gets quick and frequent feedback and information on these aspects, and are able to respond immediately to any incident related to them.

However, in the non-functional part, the situation is not so clear. There are certain kinds of tests that have already been adopted by the development teams. Examples include aspects like multi-tenancy or data access control, for which tests are already usually developed as part of the regular component tests.

On the other hand, aspects like availability testing or disaster recovery testing, which are less of a test, are usually not activities present in the day-to-day of the development team. The availability of a service, being a measure of how the service behaves over time, becomes a mark on the service monitoring activities rather than a development task. And disaster recovery requires a complete contingency plan that rarely fits the development team's tasks. These kinds of "tests" cannot be easily integrated into a development workflow.

Others, like security and performance testing, are in an intermediate adoption stage.

Regarding security, it is now relatively common that development teams integrate a static code analysis tool in their Continuous Integration system, which analyzes the code based on a pre-set collection of rules looking for common errors and design flaws. But it also provides relevant information regarding possible security holes or weaknesses that appear or can be inferred directly from the source code. However, this is only a small part of what a comprehensive security testing suite should be. Penetration testing, system security scanning, and auditing, along with ethical hacking among others, should also be part of this process. However, most of these activities are still manual and cannot be better automated as part of a development workflow.

Finally, talking about performance testing, it turns out that it's a kind of testing which is all too often left for later stages of the development, taken as an afterthought, and even left to users for validation. This is a practice that, when applied to a cloud-based application, can severely damage the image of both the application and the company behind it. The main reasons for this are usually:

- the need for having parts of the system developed.
- the difficulty of generating enough load on the system based on human testers alone.

Obviously, a functional system is needed to be able to actually perform this test. But then again, following modern development methodologies, that occurs during the first stages. On the other hand, there are available tools that help create the number of virtual users (i.e. bots) needed to generate load on the system. These are scripted sequences of steps intended to mimic the behavior of a real user (or maybe just to make some calls to an API). The funny thing about this is that once there is a script to simulate one user, it's just a matter of configuring the script execution to generate the needed load for the test.

SmartCLIDE's Take

The **SmartCLIDE** project seeks to foster and promote modern agile methodologies. The aim is to democratize ownership of testing as well as software Quality Assurance among the whole development team, making everyone responsible for the quality of their own developments. With that in mind, we are building a tool that will provide developers with plenty of utilities to build high quality software, offering the best support for each stage of development. Being a cloud-based tool, and looking specifically after cloud-oriented developments, we have put special extra attention on some critical points. According to the analysis presented earlier, testing is supported right from the process definition stage, going through unit test generation and code recommendations, all the way up to code analysis and deployment.

Since that is quite common among IDEs nowadays, we wanted to go one step further by helping to integrate some of those activities related to non-functional requirements of cloud offerings. So, in SmartCLIDE, developers will find the following interesting features:

- Security analysis integration, including reporting of metrics, weaknesses detected, and improvement points.
- Performance testing integration, including a test generator that helps create the test suite.
- Technical debt cost analysis, to inform about the estimated cost of fixing the detected technical debt left in the code.
- Deployment cost estimation, to help decide whether a cloud provider is, economically speaking, a suite for the needs.

SmartCLIDE

About SmartCLIDE

The SmartCLIDE project enables organizations on the path to digitalization to accelerate the creation and adoption of Cloud solutions.

The innovative smart cloud-native development environment will support creators of cloud services in the discovery, creation, composition, testing, and deployment of full-stack data-centered services and applications in the cloud.

This project has received funding from the European Union's Horizon 2020 research and Innovation programme under grant agreement No 871177