"This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884".

Project Number: 813884

Project Acronym: Lowcomote

Project title: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms

D3.2. Lowcomotive Integrations - Interim Version

Project GA: 813884

Project Acronym: Lowcomote

Project website: https://www.lowcomote.eu/

Project officer: Dora Horváth

Work Package: WP3

Deliverable number: D3.2

Production date: 30-11-2020

Contractual date of delivery: November 30th 2020

Actual date of delivery: November 30th 2020

Dissemination level: Public

Lead beneficiary: British Telecom Plc.

Authors:Léa Brunschwig, Felicien Ihirwe, Panagiotis Kourouklides, Joost Noppen

Contributors: Jolan Philippe

HISTORY OF CHANGES		
Version	**Publication date**	**Change**
1.0	30-11-2020	▪ Initial version

Project Abstract

Low-code development platforms (LCPD) are software development platforms on the Cloud, provided through a Platform-as a-Service model, which allow users to build completely operational applications by interacting through dynamic graphical user interfaces, visual diagrams and declarative languages. They address the need of non-programmers to develop personalised software, and focus on their domain expertise instead of implementation requirements.

Lowcomote will train a generation of experts that will upgrade the current trend of LCPDs to a new paradigm, Low-code Engineering Platforms (LCEPs). LCEPs will be open, allowing to integrate heterogeneous engineering tools, interoperable, allowing for cross-platform engineering, scalable, supporting very large engineering models and social networks of developers, smart, simplifying the development for citizen developers by machine learning and recommendation techniques. This will be achieved by injecting in LCDPs the theoretical and technical framework defined by recent research in Model Driven Engineering (MDE), augmented with Cloud Computing and Machine Learning techniques. This is possible today thanks to recent breakthroughs in scalability of MDE performed in the EC FP7 research project MONDO, led by Lowcomote partners.

The 48-month Lowcomote project will train the first European generation of skilled professionals in LCEPs. The 15 future scientists will benefit from an original training and research programme merging competencies and knowledge from 5 highly recognised academic institutions and 9 large and small industries of several domains. Co-supervision from both sectors is a promising process to facilitate agility of our future professionals between the academic and industrial world.

Executive summary

This document introduces an evolving design and implementation for a low-code development platform (LCDP) that can support low-code development activities in a range of application domains. The platform is intended to provide the base building blocks for creating specialised LCDPs in specific domains ranging from chatbots and data science to the Internet of Things. In this deliverable we report on the progress that has been made towards this goal and the future activities that are planned.

Table of contents

1. Introduction

The present document is a deliverable of the Lowcomote project (Grant Agreement n°813884), funded by the European Commission Research Executive Agency (REA), under the Innovative Training Networks Programme of the Marie Sklodowska Curie Actions (H2020-MSCA-ITN-2018). The purpose of this document is to provide an overview of the work towards app creation and integration as part of the Lowcomote research projects.

A core goal of the Lowcomote project is to explore the opportunities and potential of low-code development platforms (LCDP) in a variety of application domains. Among others the project explores domains ranging from chatbots and data science to the Internet of Things. Each of the areas naturally will cover domain specific concepts, workflows and challenges, but at the same time the Lowcomote project will explore common elements in these domains so that they can be supported by a uniform LCDP which in turn can be extended to support these specific domains. In this deliverable we report on the progress that has been made towards this goal and the future activities that are planned.

In this deliverable we will first cover the progress that has been made in defining approaches and driving towards base architectures for LCDPs within specific domains covered by the Lowcomote project and workpackages. This will cover progress in the domains of smart cities, data science and mobile applications. Second we will cover the overlap between these initial efforts and how this is shaping the future work that will lead to a uniform LDCP with integrations across various application domains.

2. Application Creation using Domain Specific Languages

A major focal point of the Lowcomote project is to explore how low code development can support the creation, deployment and management of applications. With an increasing number of roles in research and development in industry requiring some form of software development, there is a need for tools that can aid citizen developers to achieve their goals without the need for retraining. Given the specifics of the domains in which these roles are emerging, such as data science, they tend to have specific workflows, concepts and deliverables that differ from traditional software development. This makes them well suited for the application of domain specific languages (DSLs) and code generation, as these can bridge the gap between the domain and the realisation of its applications in software. In this section we cover the progress made in the Lowcomote project on the definition of DSLs and workflows for the collaborative mobile apps and data science domains.

Collaborative Mobile Apps Using Active DSLs

This Section concerns task 3.5 and its progression for deliverable 3.2. We will first remind the goals of task 3.5 and give some background. Then, we will present the progress and first results obtained during the first year of research.

Task 3.5 “Low-code Development of Rich Collaborative Mobile Apps using Active DSLs” aims at :

- Defining the state of the art of the use of modelling inside the mobile devices and the classification of these tools,
- Designing different DSLs to extend the notion of Active DSLs, including context, role-based access control and augmented reality,
- Implementing an app-based and web-based editor for the creation of Active DSLs,
- Integrating the solution inside the common LCDP of the Lowcomote project.

As a first contribution, we are realizing a Systematic Mapping Studies (SMS) [1] on domain-specific modelling with mobile devices which consists of retrieving papers from several databases (Scopus, IEEE, Springer Link, ACM, …), reviewing them and classifying the tools from these papers and comparing them.

Concerning the second goal of Task 3.5, we will first provide some background and then explain our contributions.

Traditionally, modelling is an activity which takes place in static environments, such as desktops and laptops. Moving this exercise to mobile devices will allow the exploitation of features proper to these mobile devices, for instance, geo-positioning, external interactions, context, … Overcoming these challenges will also provide a simplified environment where non-programmers will be able to model wherever they want.

The approach will use Active DSLs (Figure 2.1) for its implementation. An Active DSL is a novel approach, proposed in [2], which can be deployed on both desktop computers and mobile devices, however, the latter ones permit exploiting features which are only relevant in mobility. An Active DSL is composed of a set of “feature metamodels” that will annotate a domain metamodel created by the user. They can have any concrete syntax but we advocate the use of a

graphical syntax to represent the models in order to facilitate their use by end-users.

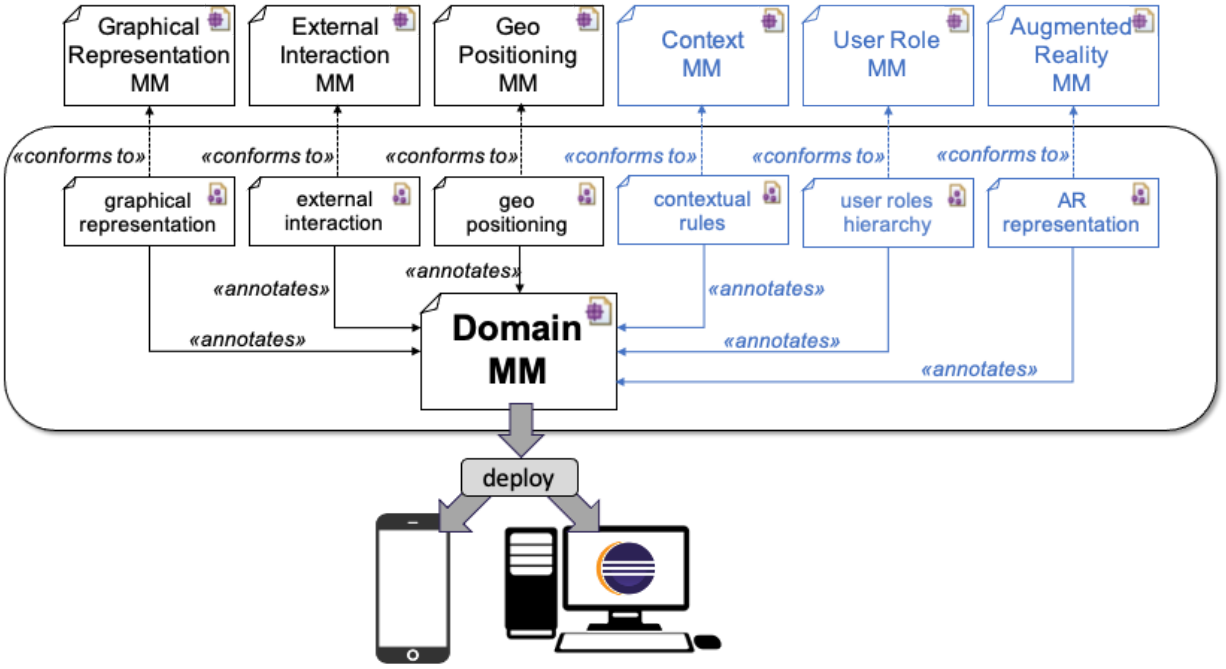

Figure 2.1: Elements to describe and deploy Active DSLs.

Active DSLs can profit from geolocation of some model elements in a map and can represent the DSL users in the model to identify their current position thanks to a geo-positioning meta-model. Active DSLs also have the ability to communicate with external services using the external interaction meta-model which considers two kinds of external elements: external devices (e.g. IoT devices) and services (e.g. web services). A context meta-model would permit to define contextual rules for rendering the domain meta-model context-aware. This would give the ability to an Active DSL to react to external events triggered by external interactions (like IoT devices or web APIs) or to the current state of native functionalities of the mobile device (like sensors or internal clock). In certain circumstances, it might be necessary to render the model or the user interaction differently depending on the roles and permissions granted to the users. Finally, the domain meta-model could also be represented using augmented reality for enhancing the users' experience. In addition to all these aspects of the Active DSLs, many modelling scenarios could greatly benefit from collaboration, either local or centralized (e.g. relies on servers and allows remote collaborations between distant users) and can enable extended modelling, which refers to additional elements beyond those described in the language, such as pictures, sketches or text annotations that enrich the collaboration or the semantic of a model.

Our second contribution has been described in [3] and we will give a short overview. We chose to focus firstly on the User Role DSL. Its meta-model is shown in Figure 2.2 and an example of its concrete syntax is Listing 2.1. It permits to describe role hierarchies and permissions to restrict the access and use of an Active DSL and its instances. We distinguish the following kind of roles:

- *Admin* which refers to the author of the language and any granted users. Every Active DSL must include exactly one role of this kind, which gives access to all functionalities and elements of the language.

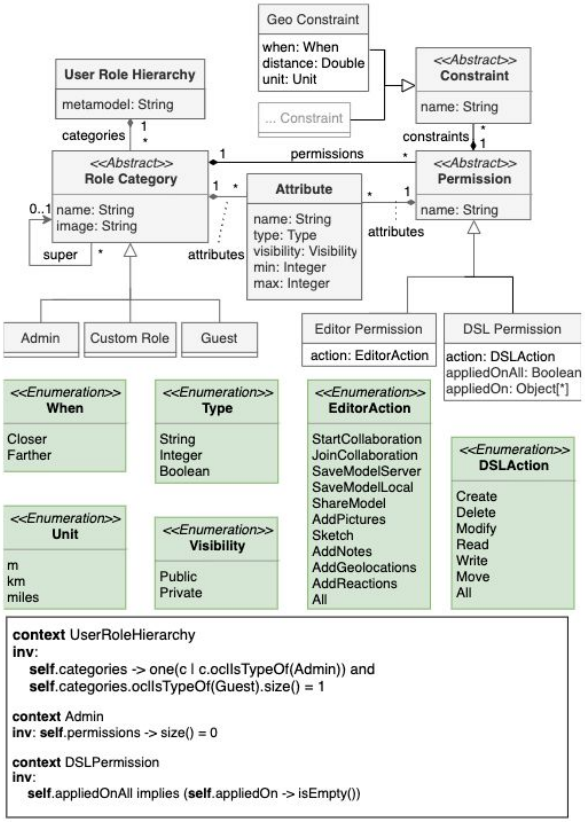

Figure 2.2: User role meta-model.

```
metamodel: "http://miso.es/dsls/tourismDSL.ecore"

  Admin {
    image: "http://www.miso.es/images/miso.png"
  }

  Guest{}

  tourist {
    // attributes
    + string name
    - integer age
    // permissions
    @DSL canRead (read) {
      -> tourism.Cultural
      -> tourism.Transportation
      //...
    }
    @DSL giveOpinion (write) {
      -> tourism.Opinion
    }
    @DSL changeOpinion (modify) {
      -> tourism.Opinion.rate
      -> tourism.Opinion.opinion
    }
    @Editor canAddPictures (AddPictures)
             when closer 10 m
  }

  touristicguide extends tourist {
    // attributes
    + string[*] spokenLanguages
    // permissions
    @DSL touristicGuideCanDoAll(all) {
      -> tourism.Museum
      //...
    }
    //...
  }
```

Listing 2.1: Example of the User Role MM concrete syntax.

- *Guest* which refers to users who have not been assigned any other role. This is typically the role with the least permissions, and by default, it forbids access to the model. There is exactly one role of this type.
- *Custom Role* which refers to language-specific roles ("tourist" and "touristic guide" in Listing 2.1).

Likewise, permissions are classified into two distinct groups which determine their action scope:

- *Editor permissions* are applied on functionalities of the editor e.g., allowing collaboration, model sharing, attaching pictures or reactions (visual alerts) to model elements, among others ("@DSL" in Listing 2.1).
- *DSL permissions* concern the management (i.e., creation, deletion, modification, etc) of the elements of the domain meta-model. They can target classes as well as their attributes and references ("@Editor" in Listing 2.1).

Permissions and role categories can define attributes to be informed when the language is deployed. As an example of the former, one could define the address or the name of an external service or device, to restrict its exploitation by users. As an example of the latter, one could define that tourists have a name and age, and touristic guides speak a number of languages. The value of this attribute should be informed for each particular user having the role. Attributes so defined can be either public or private, which means a public attribute can be seen by all but a private one can only be seen by the role categories above. The attributes of a guest can be seen by all but the private attribute of an admin cannot be seen by anyone.

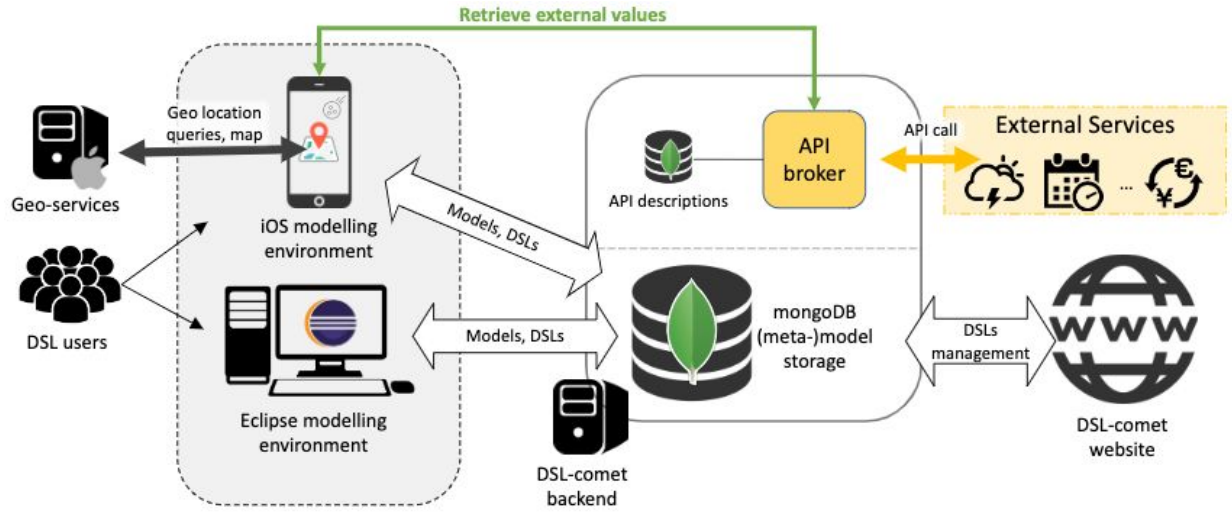

Figure 2.3: Architecture of DSL-comet.

Finally, permissions can have contextual constraints for limiting them in certain situations. In Figure 2.2, the class GeoConstraint is used in Listing 2.1 at line 27 and specifies that a tourist can add pictures only when they are within 10 metres of a location. The expansion of our role model with more contextual constraints will be the topic of future work with the creation of a Context DSL for defining contextual rules.

As depicted in Figure 2.1 and stated in the second goal of task 3.5, the notion of Active DSLs will be enriched with Augmented Reality (AR). We are currently working on the elaboration of an AR DSL and making tests.

The third goal of Task 3.5 concerns the implementation of an app-based and web-based editors for the creation of Active DSLs. We already have the former one which is called DSL-comet and we are working on the latter one.

DSL-comet is a modelling editor aimed at supporting Active DSLs and it runs on iOS devices (iPhones and iPads). It can be installed from Apple's app store, its home page is https://diagrameditorserver.herokuapp.com and a demonstration video to illustrate some of its features is available at https://youtu.be/rzhl9yMFSxI. Figure 2.3 depicts the architecture of DSL-comet. Users can download the Active DSL definitions stored on a remote database, then start building models and store them either locally or in the database. DSL-comet provides an API broker service to manage interaction with external services and supports geo-services: models can be geo-positioned on the map and it can perform geolocation queries. DSL-comet website is used for managing the list of Active DSLs and their instances.

Figure 2.4 shows the architecture of our tool with the integration of the User Role DSL and some parts are still under development. On DSL-comet's app, users can consult their current role as well as the list of roles available for a specific meta-model however the behavior of roles and permissions needs to be implemented. On DSL-comet's website, the sign in and sign up functionalities have been added, the notion of user session, roles and user role models have been added to the database, and hierarchy of users and attribution of roles for a specific meta-model is possible.

As a future work, we plan to finish the implementation of the User Role DSL in DSL-comet and to develop an Xtext editor in the cloud for defining the annotation model of the Active DSLs such as our User Role models.

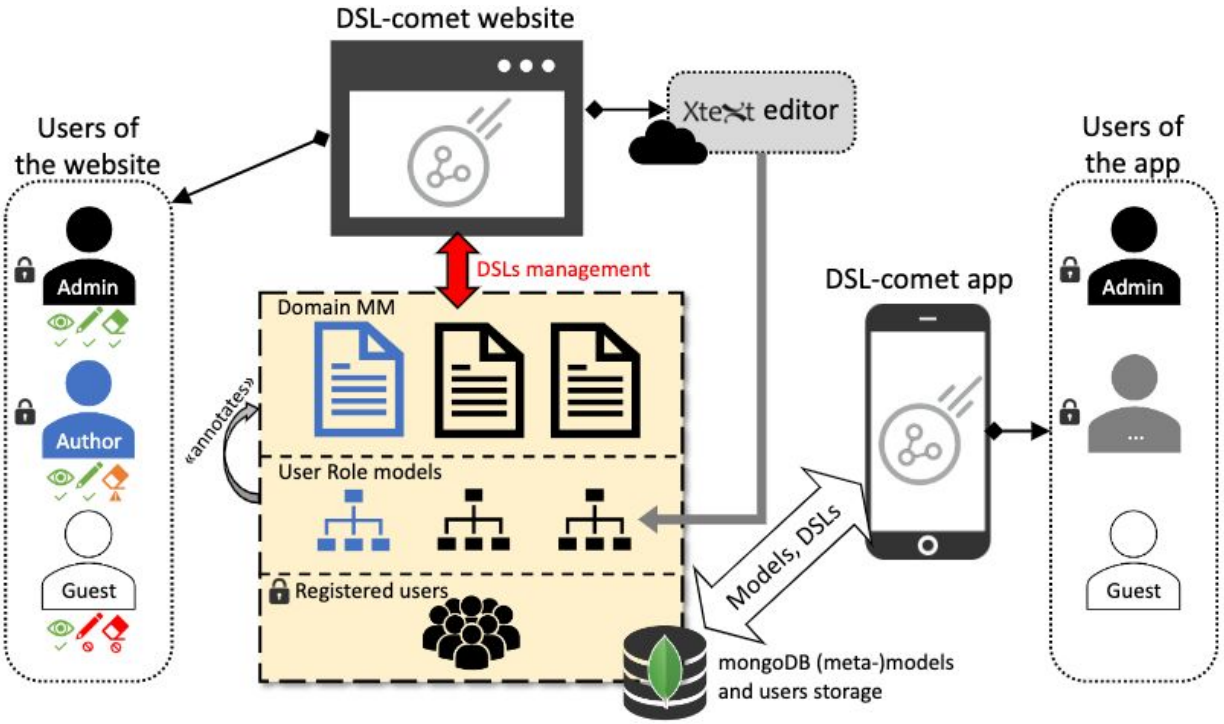

Figure 2.4: Management of roles within DSL-comet.

The ultimate goal of task 3.5 “Integrating the solution inside the common LCDP of the Lowcomote project” is not part of deliverable 3.2 and will be tackled later.

[1] Petersen, K, Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessments in Software Engineering, EASE’08, p. 68-77. BCS Learning and Development Ltd., Swindon, GBR (2008)

[2] Vaquero-Melchor, D., Palomares, J., Guerra, E., de Lara, J.: Active domain-specific languages: Making every mobile user a modeller. In: Proc. ACM/IEEE MODELS, pp. 75-82. IEEE Comp. Soc. (2017)

[3] Brunschwig, L., Guerra, E., de Lara, J.: Towards access control for collaborative modelling apps. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (2020)

Deploying and Knowledge Model Monitoring

In this section, the progress of task 3.3 in relation to deliverable 3.2 is presented. The contents of this section are going to be organized as follows: First, a short description of the goals for task 3.3, as they were more clearly defined during the first year of research, are given. Following that, we are going to present the research direction chosen as well as the progress so far in relation to those goals. During the first year of research, we published a workshop paper as part of the Models 2020 conference. Since that paper also outlined our first-year results and our future research direction, parts of it are adapted for the purposes of this report.

The overarching goals of task 3.3 is to enable data scientists to easily and rapidly deploy machine learning models in a production-grade capacity. In order to be able to achieve this goal effectively, a lot of effort was invested in getting acquainted with the intricacies of the machine learning workflow. We achieved this by following the two-fold approach of conducting a review of the available academic literature in the area, supplemented by also conducting one-to-one interviews with data science practitioners inside BT. As a result, we discovered that the machine learning workflow consists of three, equally important, parts. Namely, data preparation, model creation and the post-creation activities. In the figure below, one can see a graphical representation of the workflow divided in the three sections that we are going to discuss. For our chosen research direction to be better understood, we will now give a short description for each of the parts.

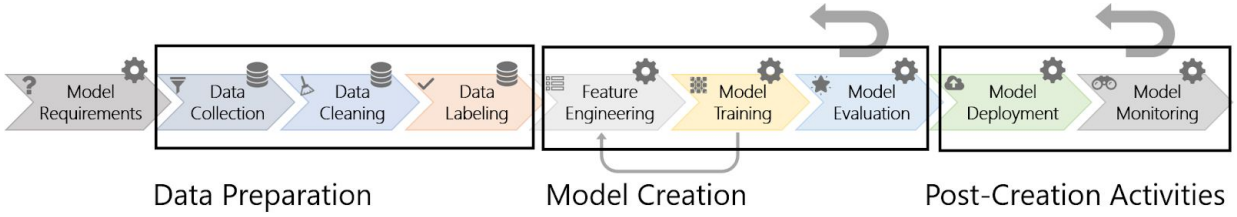

Figure 1: Typical Machine Learning Workflow

The first part of the machine learning workflow, as mentioned, is data preparation. In contrast to classical computer science algorithms, machine learning solutions need vast amounts of data in order to be effective. Thus, a large portion of the machine learning workflow has to do with data-related activities. According to an internal survey conducted by Microsoft, the top ranked challenge in the field of machine learning, as perceived by employees working in it, is data availability, collection, cleaning, and management. Due to the importance of data, organisations that integrate machine learning in their products, have invested substantial engineering resources in developing systems that can validate incoming data. These organisations have also developed systems which enable users to specify the expected properties of the data that is to be received. Subsequently, the system automatically checks that the incoming data adheres to the specifications. The goal is to keep the quality of incoming data consistent and alert the engineers of any anomalies. By ensuring that all the data is of high quality, it can then safely be used for training new models or fed into existing models for inference. In that regard, one could draw a parallel to software testing. In the same way, developers of traditional software products would like to test new code to make sure it doesn't introduce bugs in their code base, developers of machine learning systems should test new incoming data to make sure that they are consistent with what the system expects to receive.

The second group of activities in the machine learning workflow, has to do with the creation of the model artefact. These are perhaps the activities mostly perceived to be the core of machine

learning but are usually just a small part of the overall system, in terms of code volume. The model created at this stage is the result of an iterative process by which different aspects of the raw data, algorithms and input parameters for these algorithms are experimented with in order to find out which combination delivers the best performance. Keeping in mind that this experimentation might take place over several days and be performed collaboratively by several data scientists, it becomes evident that there is a need for a system that can keep track of the performance metrics of every combination of factors that was attempted. Such systems have indeed been designed and are in use by teams that train machine learning models. Every time a machine learning model is trained, the system stores a variety of metadata in a database for future reference. The stored metadata can include the dataset used for training and evaluating the model, the performance metrics achieved by the model when evaluated, the input parameters used for the training of the model and also custom fields that the developer wants to associate with that particular training run. These kinds of metadata databases are usually private, and access is given selectively to members of a specific team or company. One exception to this is the OpenML database which largely shares the same capabilities as described above but is open for anyone to contribute with new datasets, training runs etc.

The final group of activities in the machine learning workflow, are the activities that follow the creation of the model artifact. In a research setting, the machine learning workflow of the researcher commonly concludes with the training and evaluation of the model. If the produced model would achieve performance metrics superior to those of the state of the art, the results would be published for other researchers to be informed and build upon them. On the other hand, when applying machine learning techniques in a commercial setting, the goal is to incorporate the output of the model in a product that customers (internal or external) will use. For this reason, the model artefact will have to be somehow deployed so that it is reachable by the customer-facing part of the application. There are various tools that can facilitate the serving of machine learning models, such as TensorFlow-serving. In addition to deploying the model, a strategy needs to be devised for the monitoring of the model's performance over the course of time. There is no guarantee that the model's performance during its deployment will be consistent with its performance during evaluation. There are various reasons for that, such as, training based on non-representative samples, as well as the dynamic and ever-changing nature of the world. For this reason, it is essential that the performance of the model is monitored as to ensure the quality of its output on a continuous basis.

After reviewing all the aforementioned parts of the machine learning workflow, along with the relevant software tools that are available for them, we decided to focus on the area of post-model-creation activities. Specifically, we aspire to create a low-code solution that comprehensively tackles the challenge of monitoring the performance of a deployed machine learning model on a continuous basis. After researching the factors that affect the long-term performance of machine learning models, we have outlined the specific features that our low-code solution should include. Specifically, our solution will seek to automate the aspects of data capture, performance degradation detection as well as performance degradation response. To date, we have produced several demos that detect performance degradation using existing technologies, in order to better understand the gaps that exist. More importantly, we are in the process of developing the first iteration of a DSL that will tackle the problem of data capture by automatically generating all the software infrastructure that is needed, based on the data scientist's declarative statements. We believe that the successful development of such a solution will enable all data scientists to focus more of their resources in doing what they are best at, while we take care of mundane technical details.

3. Integrations for IoT for Smart Cities and Knowledge Models

In this Section we cover progress that has been made in a more specialised and comprehensive domain for low code environments, that of Internet of Things for Smart Cities. Naturally this domain has a focus on application development for such smart cities. However it also has a strong interplay between data collection devices, analysis, networks, etc. This in turn means that research in this area is driving the creation of platform elements that can address software and application development as well as hardware interaction and connectivity. The progress in this area can be found below.

Urban Area Management in Smart Cities

This Section contains the progress work under WP3: Low-code Engineering of Large-Scale Heterogeneous Systems. The task which is associated with this is Task 3.4 : Urban Area Management in Smart Cities and its deliverable would be D3.2: Lowcomotive Integrations. First we will define the goals of task 3.4 and give the topic background. After, we will present the current progress and the results so far and finally, introduce our proposed approach.

The task 3.4 entitled *Urban Area Management in Smart Cities* goal is to introduce Lowcomotive integration of IoT for Smart Cities. As a main contribution to the WP3, it will propose a domain-specific language for the design of large and complex IoT systems like Smart Cities and present the possible integration mechanisms with Lowcomotive platform.

State of the art

Nowadays, IoT is regarded as a collection of automated procedures and data, integrated with heterogeneous entities (hardware, software, and personnel) that interact with each other and with their environment to reach common goals [1]. In our daily life, we see more intelligent traffic lights, advanced parking technologies, smart homes, and intelligent cargo movement. This is due to the rising adoption of artificial intelligence (AI) and 5G infrastructure in helping the global IoT market register an increased growth. Developing such systems has to cope with several challenges mainly because of the heterogeneity of the involved sub-systems and components including human being intervention.

Figure 1 illustrates the high-level architecture of a typical IoT system. A *thing* is a combination of on-board devices including *sensors*, *tags*, *actuators*, and *physical entities* like cars, watches, etc. Data is generated from a sensor or a tag attached to the physical entity the user is interested in. A computing *device* (such as an Arduino, a Raspberry Pi, etc.) collects data and sends them to the nearby *gateway* using some well-known protocols such as Z-Wave, MQTT, HTTP, Bluetooth, Wi-Fi, Zigbee, etc.

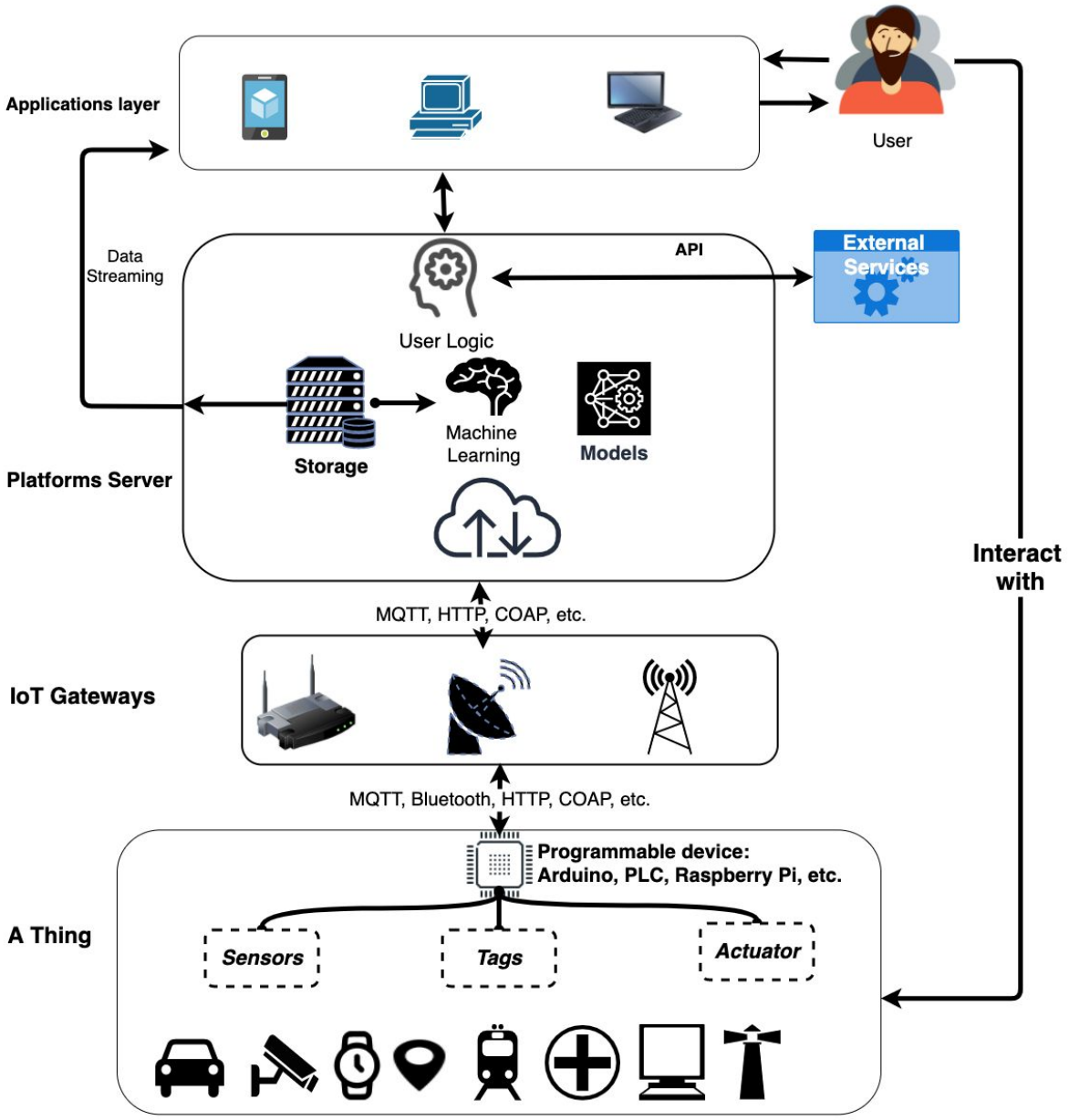

Fig 3.1. High-level architecture of a typical IoT system

The *Gateway* component acts as a bridge between the physical and digital worlds. Note that in some cases, devices and gateways can make some simple logical computation and respond to some events without the need for further processing. The platform server is a combination of processing and storage resources on the cloud. At this stage, data can be streamed, analyzed, or manipulated for meaningful information to be communicated back to things, users, or third parties services

In our first year of research, we focused on understanding the current state of the art Low-code engineering by conceiving languages and tools supporting the development of IoT systems. In our published paper at the 1st Low-code workshop at MODELS'20 conference[2], we have examined the current state of art on model driven engineering approaches for IoT by taking into account low-code development platforms in particular. The selection process was done

iteratively by exploring different platforms such as Google scholar. After some manual paper mappings we selected 16 approaches considering the fact that a given approach has a supporting tool accompanied, is more recent that 2010, uses MDE as an underlying technique, and has at least three scientific publications or reports referring to it.

The platforms were divided into two categories considering their basic implementation mechanisms. In particular, the first category consists of tools based on the Eclipse technologies such as Eclipse Modeling Framework (EMF), Graphical Modeling Framework (GMF), and Papyrus environment. The second category is a collection of tailor-made low-code development platforms. Figure 2 shows the common categories pattern based on their underlying infrastructures.

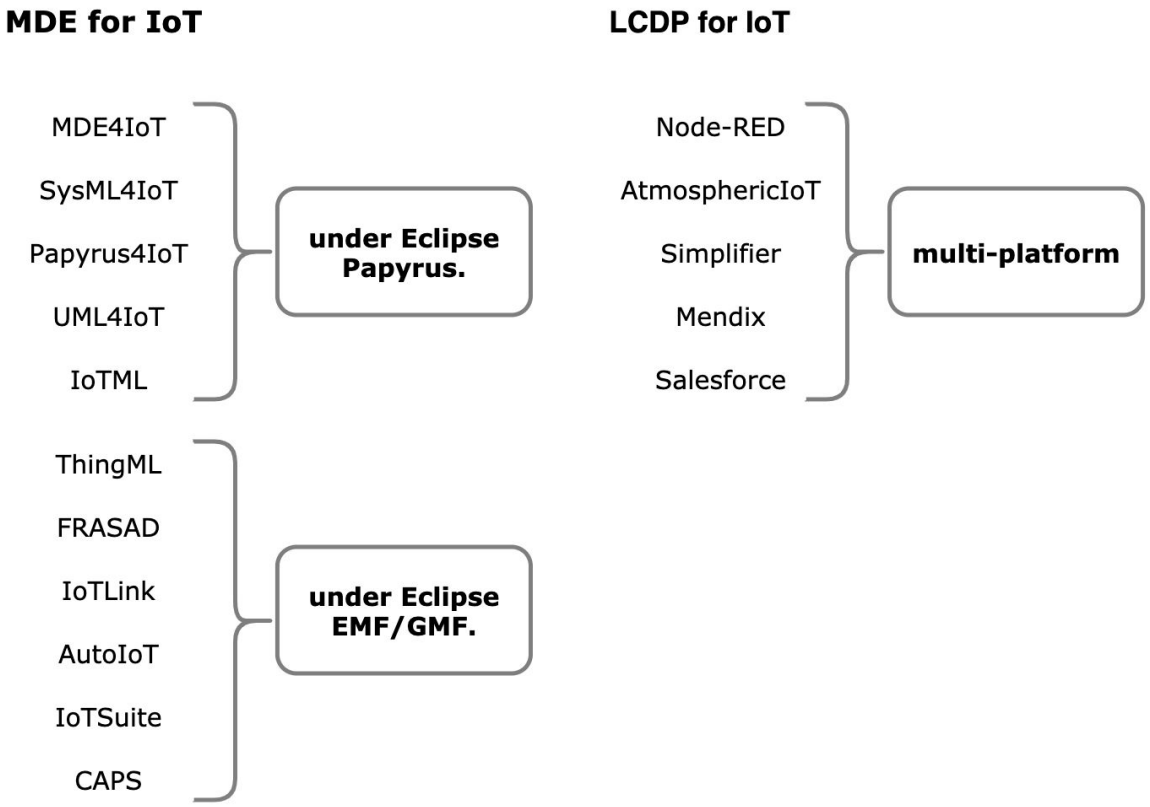

Fig 3.2. MDE and LCDP for IoT categories.

Taxonomy development and findings

Our study has been performed by conceiving a taxonomy consisting of features characterizing the studied IoT development platforms. By analyzing the languages and tools overviewed in the previous Section, we identified and formalized their corresponding variabilities and commonalities in terms of a feature diagram. These features were selected mainly based on the common understanding regarding the stages to be followed in the software development process, from requirement definition, system design, development, deployment and maintenance of a robust complex system.

Concerning the selected features we have: *Requirement modeling support* features group which evaluates whether a tool has an inbuilt requirement specification environment. Supporting this feature is very important because it helps keep track of whether the specified requirements are correctly implemented throughout the whole development. This also helps in requirements traceability and verification. *Domain Modeling support* refers to the kind of modeling tools interface the tool has, e.g., if it is graphical or not, if it gives the possibility to model the static structure of system's blocks or components. Some of the analyzed systems provide modelers with behavior modeling capabilities to specify semantic concepts relating to how the system behaves and interacts with other entities (users or other systems).

*Testing and verification suppor*t refers to whether a tool has inbuilt mechanisms to test artifacts before deployment which can be done by conducting different verification. To be more specific this feature examines if the tool has a testing workbench, an inbuilt model checking and validation facility. *Analysis environment* features are related to the capability of the considered environment to support different analysis methods for the intended system before its deployment. This can be done on different blocks or components of the system by checking on their responsiveness in case of failure, networkloss, security breach, and so on. In this regard, we can feature dependability analysis, real-time analysis, and system quality of service in general.

Reusability feature group illustrates whether the tool under analysis allows the export of artifacts for future reuse. This can be done on developed models or on generated artifacts. Reusability Features are also dealt with the way artifacts are managed e.g., locally or by means of some cloud infrastructure. *Deployment* support features evaluate the ways the developed artifacts are deployed and how ready are to be deployed. To the best of our knowledge, this should be one of the important features to focus on when implementing a novel tool. We also looked at whether the development tool can be installed locally or on the cloud depending on the client's interest.

In addition, the *Interoperability* feature examines the ability of a tool to exchange information either internally between components, exposeor consume functionalities or information from external services e.g., by means of dedicated APIs. *Extensibility* feature checks whether the tool provides the means for refining or extending the provided functionalities. In the case of modeling tools, such a feature is related to the possibility of adding new modeling features and notations. Finally, *Target support* feature refers to the characteristics of the target infrastructure, which enables the execution of the modeled system. Figure 3 shows the top-level feature diagram, where each subnode represents a major point of variation.

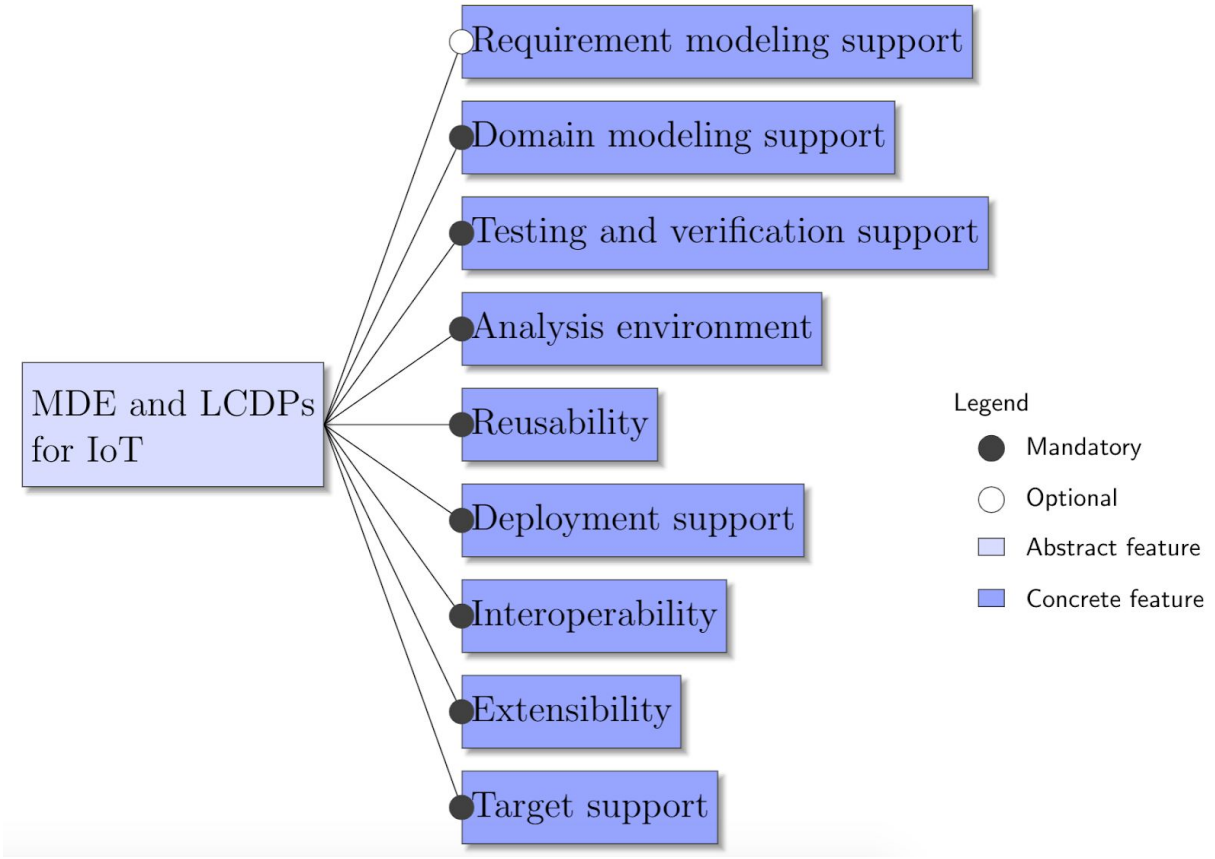

Figure 3.3: the top-level feature diagram

Further than the previous features, we included an "*additional characteristics*" field to highlight the additional aspects of the selected tools. In particular, some tools target early phases of development like system design, data acquisition, system analysis by focusing on the thing behaviour. Some other tools target the application layer functionalities without taking much care of the data acquisition phases. This is being done by integrating the tool with already implemented data source engines, etc. Another peculiar aspect is if the considered approach is available as open source or not having this an important impact on the possibility for the community to contribute to its development.

Findings

The features explained above were used to evaluate the functionalities and the services supported by each analyzed platform. Table 1 gives expanded details about the findings from the described taxonomy. In this section, we are going to list a brief description of the finding in accordance with what we've found during our study. As a last step, we identified some weaknesses and limitations of those already existing approaches and discussed possible ways for addressing them in the future.

First, according to Table 1, we can see a huge lack of focus on requirement specification except for tools such as SysML4IoT[1] (as it extends SysML[1] infrastructure which enforces requirement specification) and FRASAD[4], which enforces the requirement specification at the initial modeling phase using rules that can be tracked throughout. The huge lack of analysis support for almost all the tools selected is alarming. We think that it is highly important to analyze and verify the intended system's behavior before deployment as it gives developer indications of what may happen before deployment and helps make any adjustment earlier enough. Moreover, from Table 1 we see that most of the tools can run locally, especially eclipse-based tools; however most LCDPs can run both locally and on the cloud.

For instance, we noticed a lack of standards to support the model-based development of IoT systems due to technology evolvement and the presence of heterogeneous players making the IoT reference meta-modeling convoluted.

Second, we noticed a limited support of multi-view modeling except for CAPS[5], MED4IoT[6], AtmosphereIoT[2], and Mendix[3]. This technique presents enormous benefits as it enforces separation of concerns: the system component is designed using a single model with dedicated consistent views, which are specialized projections of the system in specific dimensions of interest [3]. Moreover, this technique is regarded as a complicated matter to address for tailor-made low-code development platforms as they mostly focus on connecting dots aiming at having an application up and running.

Third, we noticed a limited support for cloud based model-driven engineering. Moving model management operations to the cloud and supporting modeling activities via cloud infrastructures in general is still an open subject. From our study, we noticed that mostly low-code development approaches provide the option to run tools on cloud or on-premise. This is not yet the case of tools based on eclipse environment which still requires local deployments.

[1] https://www.eclipse.org/papyrus/components/sysml/
[2] https://atmosphereiot.com/
[3] https://www.mendix.com/

	MDE Approaches											Lowcode development platforms				
	BRAIN-IoT/ IoTML	MDE4IoT	Papyrus4IoT	SysML4IoT	UML4IoT	ThingML	IoTLink	IoTSuite	FRASAD	AutoIoT	CAPS	NodeRed	Atmosphere! oT	Simplifier	Mendix	Salesforce
Requirement modeling support																
Requirement specification				✓					✓					✓		
Requirement verification				✓					✓							
Domain modeling support																
Graphical user interface	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Structure model	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓	✓		✓
Behaviour model	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
Multi-view modelling	✓	✓	✓	✓	✓		✓			✓	✓	✓			✓	
Testing and verification support																
Testing environment									✓			✓		✓	✓	✓
Model checking	✓		✓	✓		✓				✓	✓					
Model validation	✓		✓			✓				✓	✓					
Analysis environment																
Realtime analysis																
Dependability				✓												
System QoS				✓							✓			✓		
Reusability																
Allow exporting artifacts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Allow importing external artifacts												✓				✓
Allow artifact storage and future reuse	✓	✓				✓		✓			✓	✓	✓	✓	✓	✓
Deployment support																
Can be deployed locally	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
Can be deployed on cloud	✓					✓						✓	✓	✓	✓	✓
Can generate code	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓
Run-time adaptation	✓	✓	✓	✓			✓	✓				✓				
Interoperability																
Support API integration					✓		✓					✓	✓	✓	✓	✓
Support connect to extern data sources	✓				✓		✓					✓	✓	✓	✓	✓
Extensibility																
Easy to add modeling features	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
Easy to modify the existing features	✓			✓		✓	✓	✓	✓	✓		✓	✓		✓	✓
Additional characteristics																
Focus on "The thing" layer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		
Focus on Application layer						✓						✓	✓	✓	✓	✓
Open source to developers	✓		✓			✓						✓				
Target support																
Underlying infrastructure	Papyrus OSGI	Papyrus	Papyrus	Papyrus, Moka	Papyrus	EMFCore Maven	EMF, GMF, EEF, Accelelo, Esper	xText, ANTLR, J2SE	EMFCore GMF	EMF, GMF	EMF,GMF, AMW, ThingML	NodeJS	Java	Node.js, Docker	Java, JS, AWS	Java, Salesforce IoT Cloud
Target platform	IoTboards, cloud servers	IoT boards	MicorEJ, Prismtech's Vortex	MicroEJ, Edison board	Contiki, Rasp. Pi	IoT Boards, OS, Cloud	Arduino	Android, J2SE platforms	TinyOS, Contiki	cloud	Same target as ThingML	platform agnostic	Mobile, Many IoTBoard	OS, Android	Cloud	Cloud
Code generation language	Java OSGI artifacts	Java, C++	Java	Java	C	C/C++, JavaScript, Java	Java	Java	C, nesC	Python, HTML, CSS, JavaScript	thingml	None	Java,c, arduaino	HTML, CSS, JavaScript	Java, JavaScript	Java

Table 1: Findings

Finally, we see a limited support for testing and analysis infrastructure. Very few tools care about the testing and analysis phases of the IoT system development process. There is still a big challenge regarding how to analyze IoT systems responsiveness before deployment. The complexity of the problem relies on the fact that IoT systems involve human interaction, environment constraints. To this point, we have also to agree on the heterogeneity of the target platforms that makes it hard to depict the kind of analysis properties to take into account.

Proposed approach

To cope with the aforementioned challenges, we are working on an ideal IoT domain-specific language named "**CHESS4IoT**" relying on existing IoT reference standards. We intend to exploit the open source CHESS tool, available on Eclipse, developed though a large collaborative R&D effort and coordinated by Intecs, due to its flexibility, reliance on standards and rich infrastructure in terms of model development, analysis and verification. CHESSML provided by CHESS, is an integrated modeling language profiled from OMG standard[4] languages such as UML, SysML, and MARTE under the Papyrus modeling environment[5].

[4] https://www.omg.org/

[5] https://www.eclipse.org/papyrus/

The ideal CHESS4IoT will be composed of a number of metamodels developed to focus on several aspects of the system. For instance to ensure the scalability and separation of concerns of our intended approach, views will be designed based on metamodel and they will be responsible of handling designs such as **requirement** definition of the system, the **structural aspects** of the system, **deployment aspects** of the developed or generated artifacts, **operational aspects** to take care of the behavioral aspects of the system by taking into account contexts and states, **functional aspects** to focus on system's functionality at runtime and finally, **communication aspects** such as network protocols and messaging mechanisms between sub-components. Figure 3.4 gives the conceptual metamodel of views of CHESS4IoT domain specific language.

This approach will uphold the multi-view technique as the aforementioned aspects will be modeled separately and merged later for transformation and artifacts generation. CHESS4IoT will be developed either by extending CHESSML metamodel with a specific set of stereotypes, contracts, and operations profiled specifically for IoT or through integrating it with new external platforms.

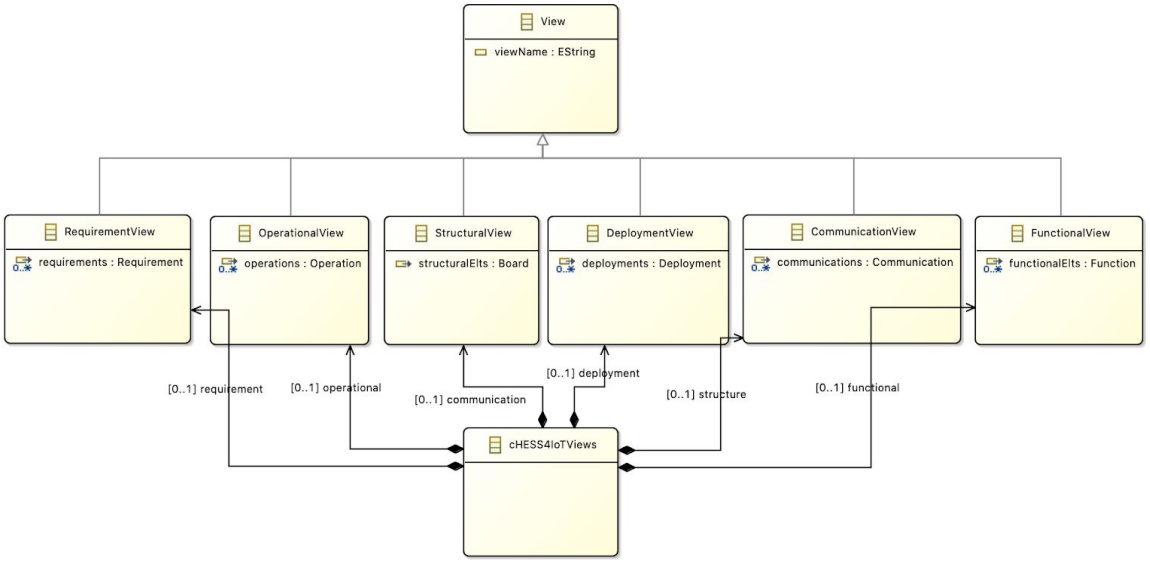

Figure 3.4. CHESS4IoT high-lever views architecture

In order to tackle the analysis and verification, CHESS4IoT will later take into account the security aspects and the already existing dependability analysis techniques such as Fault Mode Effect Analysis, Fault Logic Analysis, and Fault Tree Analysis available in CHESS. On the code generation side, we intend to use the open-source ThingML code generator because of its advanced multi-platform code generation which supports various target programming languages such as C, C++, Java, Arduino, and JavaScript. At long last, the Lowcomotive integration will be accomplished by permitting the locally developed model and generated code to be deployed to the Lowcomote repository.

REFERENCE

[1] Flávia C. Delicato Paulo F. Pires, Bruno Costa. 2016. Modeling IoT Applications with SysML4IoT. *42th Euromicro Conference on Software Engineering and Advanced Applications (2016).*

[2] Felicien Ihirwe, Davide Di Ruscio, Silvia Mazzini, Pierluigi Pierini, and Alfonso Pierantonio. 2020. Low-code engineering for internet of things: a state of research. *23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '20)*. Article 74, 1–8. DOI:https://doi.org/10.1145/3417990.3420208

[3] Laura Baracchi, Silvia Mazzini, John Favaro. 2015. A model-based approach across the IoT lifecycle for scalable and distributed smart applications. 2015 *IEEE 18th International Conference on Intelligent Transportation Systems (2015)*. https://doi.org/10.1109/ITSC.2015.33

[4] Thiago Nepomuceno, Tiago Carneiro, Paulo Henrique Maia, Muhammad Adnan, Thalyson Nepomuceno, and Alexander Martin. 2020. AutoIoT: a framework based on user-driven MDE for generating IoT applications. *In Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC '20)*. 719–728. DOI:https://doi.org/10.1145/3341105.3373873

[5] M. Sharaf, M. Abusair, H. Muccini, R. Eleiwi, Y. Shana'a, and I. Saleh. 2019. Generating Heterogeneous Codes for IoT Systems Based on CAPS. *In 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). 736–737*

[6] R. Spalazzese and F. Ciccozzi. 2016. MDE4IoT: Supporting the Internet of Things with Model-Driven Engineering. *In International Symposium on Intelligent and Distributed Computing (2016), 67–76.*

4. Summary and further developments

In this deliverable we have reported on the progress the Lowcomote project has made with respect to Low-Code Development Platforms (LCDP) in a variety of application domains. Within Lowcomote, a variety of domains is being studied within the context of LDCP, ranging from data science and mobile application to IoT and Smart Cities, and each of these domains offers unique challenges and development processes for its citizen developers. It is within the remit of this project to determine within this constellation of domains where commonalities can be found and how domain specific requirements and workflows can be supported by a uniform LDCP.

The progress reported in this deliverable focusses around the areas of collaborative mobile apps, IoT and Smart Cities, and data science, and within these domains the first concepts and domain specific language elements have been identified and described based on in-depth analysis of previous work, case studies and prototype implementations. The next steps will consist of analysing the prototypes for commonalities and variabilities and defining the uniform architecture for the Lowcomote LDCP that makes these common elements available to serve as a foundation for specialised development platforms. This includes for example uniformised code generation capabilities and DSL support. The variability analysis of these prototypes will reveal where the base platform needs to support flexible integration and extensions that allow supporting domain specific actions, workflows and artefacts. The realisation of this will be covered in subsequent project deliverables.