A Catalog-based Platform for Integrated Development of Simulation Models

Arthur Strasser[1], Peter Engel[2] and Mirco Schindler[3]
Institute for Software and Systems Engineering
Clausthal University of Technology
Clausthal, Germany
[1] Email: arthur.strasser@tu-clausthal.de
[2] Email: peter.engel@tu-clausthal.de
[3] Email: mirco.schindler@tu-clausthal.de

Wilhelm Tegethoff[4], Sebastian Lempp[5]
TLK-Thermo GmbH
Brunswick, Germany
[4] Email: w.tegethoff@tlk-thermo.com
[5] Email: s.lempp@tlk-thermo.com

***Abstract*—In the automotive domain, it is common practice to develop a vehicle system with reusable components in order to reduce development time and costs. Several suppliers are responsible for the development of the components on behalf of one leading manufacturer, who ensures the integration of the components into the system. Thereby, models are used for simulation and test of components in advance. The manufacturers integrate these models of different suppliers into their system under development using its own simulation environment. However, in order to optimize the system in a simulation, manufacturers often rely on the supplier's expert knowledge regarding components property values. But often the models must also be modified to allow their execution in a target simulation environment. Thus, manufacturers have to cope with manual steps and a decreasing re-usability of models. To overcome these difficulties, significant additional effort and costs in every development iteration is involved. A platform for automating the optimization and version management of models is a promising approach, to reduce this development effort as a common basis of the development teams. Hence, we propose a component simulation-software catalog platform for a cooperatively organized development environment. It provides a domain specific language as a meta model for modeling catalogs consisting of model variants and versions. Furthermore, the platform provides automation services for model import and export, refactoring and simulation.**

Keywords–Metamodeling; Software Ecosystem; Software Platform; Architecture Description; Simulation.

I. INTRODUCTION

In the development of electrical vehicles, manufacturers apply model-based system simulations to a great extent. In general, a simulation can be used to approximate the behavior of a system before its construction in a real world environment. In the field of Heating, Ventilation and Air Conditioning (HVAC) system development, for example, the energy saving potential of different topologies can be estimated within a simulation before starting the construction in a further development step. The development of such a system simulation is cooperatively organized. The models are usually developed by suppliers using model-based simulation tools. Then, a system manufacturer integrates these models into his own simulation environment, such as, for example, as a so-called co-simulation. Thereby, these models are coupled in an execution environment, which is different than initially planned by the design of the models. Thus, each model must be configured in such a way that it can be executed in conjunction with all other models within a third party simulation environment. As a result, manufacturers have additional expenses for software licenses and training of software developers. In addition, the interfaces of models are often modified to enable their integration into the simulation environment. But this approach hampers the reuse of models in different system simulations.

The following scenario illustrates the cooperative development of a system simulation: In order to fulfill the requirements of a future electric vehicle generation, existing subsystems of a car are further developed. For example, a manufacturer must identify the potentials for energetic savings of the next generation HVAC system. The manufacturer selects suitable components from the supplier's component catalog to develop a HVAC system simulation. However, the interfaces of the selected models can be either in a standardized format (e.g., Functional Mock-up Unit [1]) or other third party formats as Matlab or Dymola. As depicted in Figure 1, a catalog consists of hierarchies of models, which are differentiated as series and variants. For example, there are mechanically or electrically driven air conditioning compressor series. A variant from a series represents a model of a specific compressor with its specific properties (e.g., refrigerant type, discharge volume, etc.). All models are managed in version managed repositories: The further development of a model is then represented as a model version. For example, the modification of the model interface or the fix of a model error can be stored in the repository. The description of the properties of series, variants and versions is hereafter referred to as metadata. Those metadata are managed in-house by the supplier. As a consequence, the models must be selected on the basis of their metadata and configured to be compatible with each other in order to achieve energy savings. Hence, the manufacturers have to cope with the following additional effort.

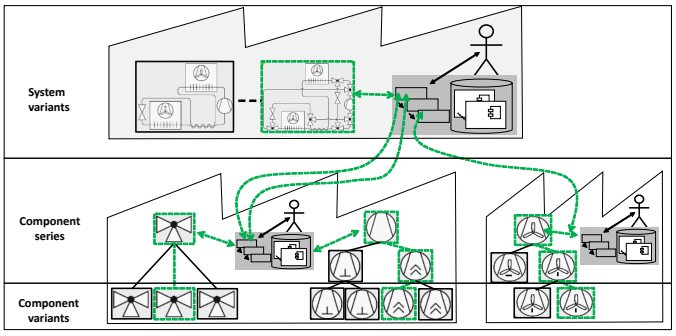

Figure 1. Current state of practice in the development of simulations models: Each developer uses his own development environment in each layer.

 ISBN: 978-1-61208-781-8

- Selection and configuration of a suitable model with regard to the individual requirements with the support of the supplier.
- To carry out the optimization and commissioning of FMUs, the manufacturer must implement ad hoc solutions that cannot be reused for the development of next-generation systems.
- Using ad hoc solutions to implement the model in the own simulation environment, which hampers the reuse of models.

In order to complete the development in a time and cost efficient manner and to ensure the reuse of models in a cooperative organized development, we propose a catalog based platform, which is described in greater detail in the following.

According to German Association of Engineers (VDI) guideline 2206 [2], a seamless tool support is necessary for a systematic system development. However, the current development practice is often not seamless, as seen for example in the development of customer-specific solutions used in the supplier's tool chain. Thus, the contribution of this paper is a proposal for a software-driven catalog platform that provides components for developers and users to support integrated development of simulation models in a systematic, cost and time-efficient way. To achieve these goals, the platform must fulfill the following targets:

- The platform must enable developers to describe metadata on the basis of a model's description language.
- The platform must enable developers to describe catalogs as compositions of components and systems.
- The platform must provide a versioning and refactoring services to increase the reusability of simulation models.
- The platform must provide model inversion techniques to ensure a maximum of usability for different development and simulation environments.
- The platform must provide services to facilitate model search, model commissioning and model changes on the basis of the catalog.

In the following, it is assumed that the exemplary models are based on the FMU format as one exemplary format for the catalog.

In Section II, the current research topics are discussed. To tackle the issues in the cooperative development, such as the HVAC systems, the concepts for the development of the catalog platform are proposed as infrastructure and as services in Section III. Afterwards, in Section IV the realization of the concepts as an overall architecture design is presented and Section V concludes from the results.

II. Related Work

In the following, we introduce the related work that addresses some aspects of our contribution. To the best of our knowledge, no overall infrastructure, and services - platform for seamless development and integration of catalog models in the field of HVAC system simulation modeling exists.

A. Frameworks for Modeling Compositions from FMUs

The creation and adaptation of simulations are development steps that belong to the composition. There are some frameworks from research approaches for the composition of FMUs.

In [3], the MOKA framework for object-oriented modeling of FMU-based CoSimulations is presented. The framework provides a language for modeling the structure of integrated FMUs based on the classes FMUBlock, FMUPort. The FMUMaster takes over the execution and instantiate the FMU blocks. An algorithm for the master-slave based execution of composed FMUs is also presented in [4]. The OMSimulatior is another FMU based modeling and simulation tool presented in [5], which provides Transmission Line Modeling connectors to enable the composition of TLM based buses using connectors. Furthermore, there are approaches to adapt the communication behavior of an FMU through wrappers. [6] presents a FMU wrapper descriptions framework for the implementation of a client-server interface. DACCOSIM [7], FMIGo [8] and FMUProxy [9] are further approaches for the distributed execution of an FMU based simulation. Another work deals with semantic adaptation to adapt the interaction for the communication of FMUs [10].

B. Merge of Simulation Models

In collaborative development processes, system variants are developed in parallel by different teams. In order to automate the integration step, an approach for the integration of ASCET-based simulation models with the Team.Mode tool is presented in [11]. The tool provides a mechanism to import ASCET models in AXL format and automatically integrates them into one ASCET model in a subsequent step.

C. Integrated Development Environment

In the automotive sector, seamless integration is known as the integration of tools using a common development environment. In [12] Broy et al. present requirements which a seamless development environment has to meet. A fundamental property of the concept is a one main repository for storing and maintaining information common to all development teams. Reichmann et al. present approaches for the implementation of this concept [13] [14].

III. Catalog Platform Infrastructure and Services

We define the catalog platform as a platform for the administration, development and versioning of the catalogs and their components. This includes the description of catalogs and modeling metadata using the catalog description language as well as services for the configuration on the basis of metadata from the catalogs. These concepts are then used for the integration of the platform into an overall architecture design for simulation development environments in Section IV.

Infrastructure. In Section III-A, we present the description language and their modeling rules. We call it infrastructure of the platform, since it is the basis for the description of catalogs and metadata as well as their composition to complex systems.

Services. The services of the platform provide additional user interfaces in order to automate the development steps

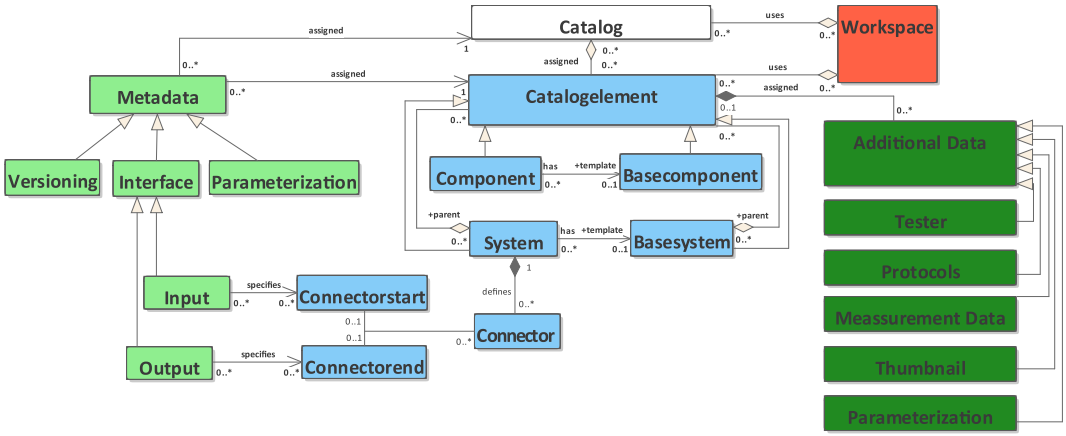

Figure 2. The Platform Meta Model for the description of workspaces and catalogs.

when reusing models from the catalog. All services are described in more detail in Sections III-B, III-C, III-D, III-E.

A. Description Technique

Catalog The catalog is the shared resource between manufacturers and suppliers in the economic market for simulation models of physical components. Using the platform, the manufacturer is able to import the catalogs of various suppliers into his development environment and is then able to create a simulation model, for example, a HVAC system. Therefore, these sections present the description language as the meta model of the platform to describe catalogs. The meta model defines the syntax and the semantics of the language, which is depicted in Figure 2. In the following paragraphs, all class names are written in italic lower case letters - clarifies the textual notation - for the explanation of the meta model from Figure 2.

Workspace The *workspace* is the working area integrated into the development environment. It can contain *catalog*s, where areas a *catalog* can contain systems and components. The language syntax defines a *workspace* that can contain any number of *catalog*s. *Catalog*s for components and for systems can then be created using the simulation environment in the *workspace*. Therefore, the meta model defines the classes *component*, *system* as subclasses of the superclass *catalogelement*.

Component A component from the catalog represents a model that can be executed in a simulation environment of a simulation tool. It declares interfaces for the communication to other *component*s. Hence, for example, an FMU or a composition of FMUs can be described by a *component*. A *catalog* that contains only *system*s and can be used as the manufacturers in-house *catalog*, which is not offered on the market to other competitors. As depicted in Figure 2, the *system* can contain any number of *component*s and *connectors*.

Connector A *connector* describes a directed point-to-point connection for the communication from one *component* to another *component* by using *connectorstart* and *connectorend*. Therefore, a *connector* must have a reference to a *connectorstart* and to a *connectorend*.

Metadata The *metadata* is defined as the superclass of different *metadata* subclasses that can be assigned to a particular *catalogelement*. The *versioning* subclass describes a unique node in a version graph for the description of further developments of *catalogelement*s and variants of *catalogelement*s (see Section III-B). An *interface* class describes the declaration of a variable with a data type from the platform. The component is the origin of a variable declaration description and hence is part of a particular *component* that uses it to define its communication to other *component*s. The *parameterization* subclass describes differential states and initial values that are required to calculate the initial conditions of a *component* that is a prerequisite to execute it in a simulation (see Section III-E).

Basecomponent To ease development of a *catalogelement* from reused *component*s, we introduce the *basecomponent*. The *basecomponent* is a *catalogelement*, that is defined only by the *interface*s from its *metadata* record. It is an abstract *catalogelement*, as it does not implement its *interface*. But, it describes a template for the development of a *component* that is expected to implement that *interface* from the *basecomponent*. For example, a new *component variant*, in addition to an existing *component variant*, can be introduced as part of a commonly shared *basecomponent*.

Basesystem A *basecomponent* can also be used to develop new composition *variant*s from existing *catalogelement*s. For that purpose, the *basesystem* is used to describe a composition from *basecomponent*s. To develop a new *system variant* from a *basesystem*, *component*s must be selected by developers that are compatible to the *basecomponent*s from a particular *basesystem*.

Constraints for valid descriptions. Developers can create correct and not correct descriptions using the description language. A description is in the set of all correct **catalog** descriptions, if it holds the following constraints:

- A *connector* must reference a *connectorstart* and a *connectorend*.
- The *interface* of a *connectorstart* and of the *connectorend* must be compatible.
- A *catalog* contains only a *basecomponent* set and a *model* set. Each *component* depends on a *basecomponent*.
- A *basesystem* must only contain a set of *basecomponent*s.

 ISBN: 978-1-61208-781-8

- The *metadata* of a *basecomponent* has an *interface* description but has no *versioning* and no *parameterization* descriptions.

B. Version and Variant Management

This section describes the version and variant management, which is a central service of the introduced platform. This is of fundamental significance, as all *catalogelement*s evolve over time. Therefore, each *catalogelement* can be assigned to a version information via the concept of *metadata*. The *version* is defined as a successor relationship: If n_i is a *catalogelement*, then $i \in$ INDEX is the version number of a given index set INDEX and n_{i+1} the successor of n_i and represented by the following relation:

$$\texttt{successorOf} := \{(x, y) \mid y \text{ is successor of } x \text{ with } x, y \in \text{CATALOGELEMENT}\} \quad (1)$$

As described in the meta model (see Figure 2) a *component* as well as a *system* can be derived from a *basecomponent* or *basesystem*. In the following, all relations are specified regarding a *component*, for *system*s the same relations are defined accordingly. This allows the implementation of variants, whereby the base element is the smallest feature shared by all variants. The variant concept is defined by the following relation:

$$\texttt{basedOn}_{\text{COMPONENT}} := \{(x, y) \mid y \text{ is derived from } x \text{ with } x \in \text{BASECOMPONENT and } y \in \text{COMPONENT}\} \quad (2)$$

If n and m without loss of generality are two *component*s and x is a *basecomponent* from which both *component*s are derived, then n and m are variants of the same *basecomponent*. Like the *component*s themselves, a *basecomponent* can of course also be versioned, generally, the `basedOn` relation can exist between concrete versions of *catalogelement*s. Each *catalogelement* can be assigned to one or more *catalog*s. There are no restrictions regarding the version or variant, e.g., a *component* can be contained in different versions or a derived *component* without its *basecomponent* in one *catalog*. The `assignedTo` relation between a *catalog* C and a *catalogelement* n is defined as follows:

$$\texttt{assignedTo} := \{(n, C) \mid n \in C \text{ with } n \in \text{CATALOGELEMENT and } C \in \text{CATALOG}\} \quad (3)$$

A successor relationship, known as a version, can also be defined between two *catalog*s. The introduction of a *catalog* as a group for versioning, distinguishes this approach from existing ones such as CVS (concurrent versions system), SVN (subversion) or git. Another special feature of common versioning systems is shown in Figure 3 by the `replacedWith` relation. This relation describes a successor relation but not in the sense of a new version, because it does not have to be a further development of the previous *component*. In practice, the most common case for this relationship is when a product is no longer supported and, as a consequence, another product has to be used instead. This is also a relation, as the `succesorOf` relation, which can be defined between two *component*s as follows:

$$\texttt{replacedWith}_{\text{COMPONENT}} := \{(x, y) \mid x \text{ is replaced with } y \text{ whereby } x,\ y \in \text{COMPONENT}\} \quad (4)$$

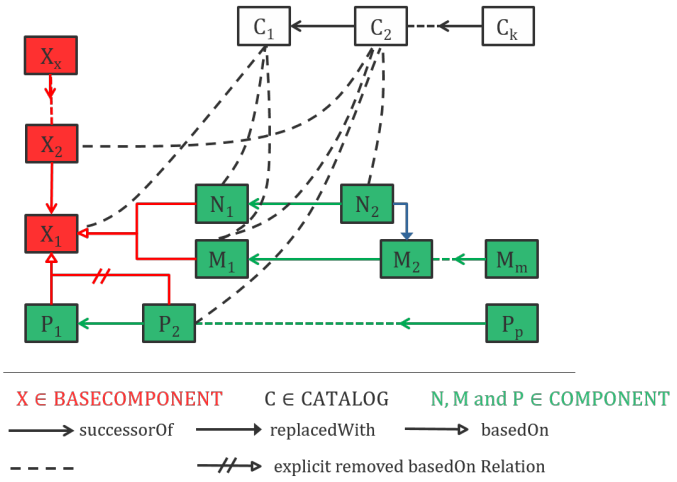

Figure 3. The versioning graph notation for the representation of the version snapshot provided by the versioning service.

As an exemplary, in Figure 3, the five specified relations between the different *catalogelement*s and the *catalog* itself are illustrated. For the purpose of simplicity not all existing or possible connections and elements have been drawn up. Another notation constituted in Figure 3 allows to illustrate a remove, as a development step, of a `basedOn` relation. This is another typical case from real practice, which is the customer-specific or prototypical development, result in the fact that a *component* is no longer part of a series.

In addition to the new types of relations described above, the challenges of version and variant management include the storage of this information and the construction of the customer-specific version graph. This is determined by the architecture of the platform, but it also offers mechanisms for a solution. Due to the fact that a customer does not have to purchase every *catalog*, it might be that the customer does not own all versions of a *catalog*, e.g., the version and variant graph has to be constructed for every platform instance. From this, the requirement for a *catalog* instance is derived. The new instance has to keep all necessary information available. This can be automated by another central service: The service provides additional or up-to-date information for the *catalog* owned by the producer. This mechanism allows that not every *version* and variant must necessarily be assigned to a *catalog*. This is also consistent with practical experience, because only certain releases of a *component* are offered to customers of *catalog*s.

C. Thumbnail Search

The common use cases with regard to data intensive or file-based systems are searching and comparing, so this is also the case with the platform introduced here. Finding and accessing of *catalogelement*s is not only, as described in this section, an important tool for human interaction. But also important as a base for the automated discovery of *component*s. For example, this automation is mandatory for the refactoring service.

To improve the search and also the comparability between components, an own variant of the so-called tag cloud [15] was developed. In general, the information stored in the *metadata* is used as a basis for the search.

 ISBN: 978-1-61208-781-8

The *metadata* are also used to generate the thumbnail shown in Figure 4, which shows two tag clouds, each visualizing series of fans. This view constitutes an overview of the stored properties and their distribution within the series at a glance. In most cases the properties are technical ones. Hence, the illustration is based on the triplet consisting of identifier, value and unit. Furthermore, the following rules are used for the generation step:

1) Elements that have the same identifier are grouped together.
2) Elements that have the same unit are grouped together.
3) The font size per group is determined as follows:
   a) The font size of an identifier is the larger the more variants exist for this identifier.
   b) The font size of a value is the larger the more elements have this value.

These rules can be adapted or extended as required. For example, the added value can be further increased. Also, semantic approaches are applicable as described in [16].

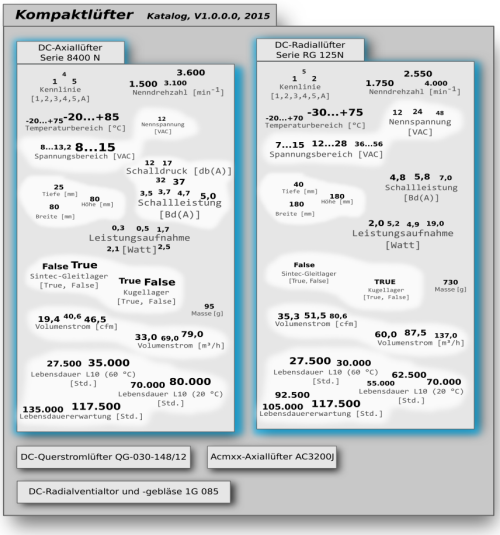

Figure 4. Thumbnail Example: Visualization of Variants

The thumbnail based search allows to create an overview even of a large number of different variants. To reduce the storage space, the thumbnails themselves are generated exclusively from the *metadata*.

D. Refactoring

The versioning of a *catalog* was introduced in Section III-B. A new *version* in the version graph describes a change to a *component* as a relation to its predecessor *version*. Customers of *component*s from that version graph must be aware of these changes, in particular critical bug fixing updates should be made available in their *catalog*s and *catalogelement* compositions, e.g., *system*s. In the following, we introduce a platform service that provides automation in the following distinct cases:

1 To replace a *component*, because it has a bug.
2 To improve an existing *component*.
3 To replace a *component* by another different *component* (The behavior of the new *component* and the original *component* need not be identical).

Using the service in the last case, can make changes to the bahavior of a simulation. Hence, notwithstanding with the general definition [17], we define *Refactoring* as a service for the automatic propagation of changes of a *component* to all *catalog*s and *system*s which use this *component*.

If a new *version* of a *component* exist, in the first case, the service replaces the *component* with the new *component version*. If no new *component version* exist, then the service informs the user to treat the error in that particular *component* manually. In the second case, the service automatically replaces the *component* by the improved *component*, if the *interface* of the improved *component* is identical to the *interface* of the predecessor *component*. In the third case, it is often necessary to replace an outdated and no longer supported *component* by a another *component*. The outdated *component* is then automatically replaced, if one of the conditions from case one or two can be applied.

E. Modelinversion

As stated in Section I, HVAC simulation *component*s can influence their performed control tasks to achieve a certain system behavior, e.g., to control the temperature of a vehicle cabin. To control the temperature to a certain operating range, an appropriate state of the simulation behavior must be reachable. Therefore, the overall simulation behavior must consider error signals of the control environment to reach the necessary state of operation called steady-state. Moreover, the control operations of the *component* must be performed in such a way that the system consumes as less energy as possible. In the first case, initialization conditions must be found for a *component*. In the second case - in addition to correct initialization conditions - a certain optimization and solution method must be applied to find an optimized solution. We introduce Modelinversion as a platform service that allows the robust and accurate as well as fast solving of algebra-differential equation systems to calculate steady-state simulation results. The calculation of simulation results for different stationary operating points is the core application of the Modelinversion.

The service forms the basis for the following applications: (1) for stationary model fitting, (2) for calculating optimal experimental designs (DoE) and (3), it offers ”numerical inversion”. Thereby, differential states, which are otherwise calculated by integration, can be specified externally using the *metadata* from the *catalog*s. The robustness, accuracy and speed of solution finding is achieved by a combination of DAE solvers and algebraic solution methods. Depending on the model, the appropriate solution method or a combination is selected: DAE solvers integrate a simulation model over time and can simulate robustly down to the steady state by methods such as flexible step size and event handling. Algebraic solvers based on a zero-point search of the state derivatives provide accurate results and are very fast in calculating many similar operating points, e.g., for different measurement points.

 ISBN: 978-1-61208-781-8

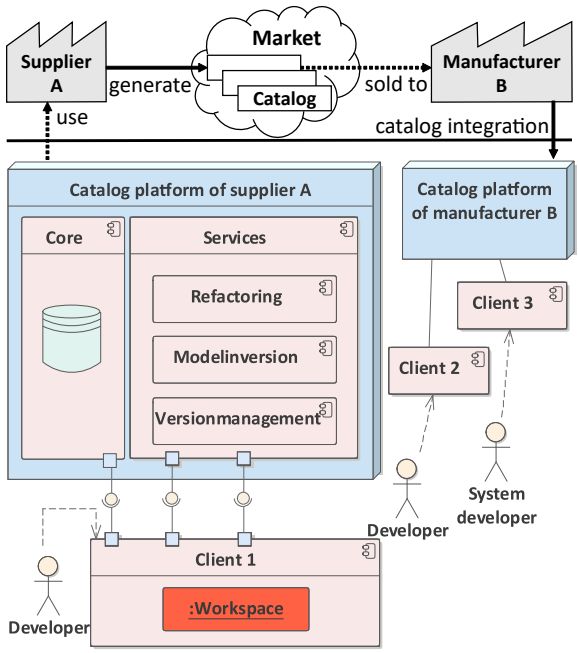

Figure 5. The overall design of the platform

IV. The Overall Context and Design of the Platform

The design and the usage context of the platform are depicted in Figure 5. The platform enables suppliers and manufacturers to exchange *catalog*s on a common market to develop simulation models based on *component*s and *system*s. The *catalog* description language from the platform defines a standardized format for the automation of the exchange tasks and simulation model commissioning tasks using platform services. This eliminates the time-consuming manual effort required to implement ad hoc solutions in the cooperatively organized development of simulation *model*s. Manufacturers can obtain the *catalog*s from the market and reuse them for the development of different *system* simulations. The *metadata* from the descriptions of the *catalog*s is used for the purchase process via a web service of a certain supplier or via a market place organized centrally by all suppliers. This enables suppliers to provide customer information about the *component*s of *catalog*s in advance without having to provide the *catalog*s themselves. From the technical point of view, the client-server architecture was selected as the architecture style for the platform: several developers can access the *catalog*s of a centrally managed repository bidirectionally and independently of each other, regardless of the development environment they use, and carry out their development locally on their client *workspace*. The clients use the graphical user-interface for developing the *catalog*s and for using the services. The platform, as the server, then manages clients access to the repository and to the platform services.

V. Conclusion

The current state of practice in the development of the HVAC *system* domain requires great expertise from *components* suppliers and additional manual handling from suppliers and *system* manufacturers to exchange and to commission simulation models in the cooperatively organized development of process *system* simulations. We proposed a *catalog* platform particularly for *component*s that are exchanged to enable seamless and integrated simulation based development process. Therefore, an infrastructure for modeling and for the exchange of *catalog*s was introduced. *Catalog*s contain *component*s and compositions from *component*s called *system*s. Both are generalized as *catalogelement*s and differentiated by *metadata* assigned to them. Thereby, the *metadata* of *catalogelement*s is used for platform services and is used to support marketplace technologies, e.g., purchase procedures. Besides the infrastructure, we proposed also a set of services provided by the platform to automate several development tasks. This proposal is part of an ongoing research, where we study upcoming use cases for the application of our platform as a future research work.

Acknowledgment

The results of this contribution are based on the work of the project “Kataloggestützte interdisziplinäre Entwurfsplattform für Elektrofahrzeuge (KISEL)”. KISEL is supported by a funding from the Federal Ministry of Education and Research of Germany in the framework of “KMU-innovativ: Informations- und Kommunikationstechnologien”.

References

[1] Modelica Association. Fmi standard.

[2] “Design methodology for mechatronic systems,” Verein Deutscher Ingenieure, Beuth Verlag GmbH, 10772 Berlin, Standard, Jun. 2004.

[3] M. Aslan, H. Oundefineduztüzün, U. Durak, and K. Taylan, “Moka: An object-oriented framework for fmi co-simulation,” in Proceedings of the Conference on Summer Computer Simulation, ser. SummerSim ’15. San Diego, CA, USA: Society for Computer Simulation International, 2015.

[4] M. U. Awais, W. Gawlik, G. De-Cillia, and P. Palensky, “Hybrid simulation using sahisim framework: a hybrid distributed simulation framework using waveform relaxation method implemented over the hla and the functional mock-up interface,” in SimuTools, 2015, pp. 273—-278.

[5] L. Ochel and et al., “Omsimulator–integrated fmi and tlm-based co-simulation with composite model editing and ssp,” in Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019, no. 157. Linköping University Electronic Press, 2019, pp. 69–78.

[6] L. I. Hatledal, A. Styve, G. Hovland, and H. Zhang, “A language and platform independent co-simulation framework based on the functional mock-up interface,” IEEE Access, vol. 7, 2019, pp. 109 328–109 339.

[7] V. Galtier and et al., “Fmi-based distributed multi-simulation with daccosim,” in Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, ser. DEVS ’15. San Diego, CA, USA: Society for Computer Simulation International, 2015, pp. 39—46.

[8] C. Lacoursière, “FMI Go! A simulation runtime environment with a client server architecture over multiple protocols,” in Linköping Electronic Conference Proceedings. LiU E-press, 2018, pp. 653–662.

[9] L. I. Hatledal, H. Zhang, A. Styve, and G. Hovland, “Fmu-proxy: A framework for distributed access to functional mock-up units,” in Proceedings of the 13th International Modelica Conference. Linköping University Electronic Press, 2019.

[10] C. Gomes and et al., “Semantic adaptation for FMI co-simulation with hierarchical simulators,” Simulation, vol. 95, no. 3, 2019.

[11] M. Janßen, C. Bartelt, and A. Rausch, “Tool-support in cooperative modeling and variantmanagment of electronic control unit software,” in INFORMATIK 2012, U. Goltz, M. Magnor, H.-J. Appelrath, H. K. Matthies, W.-T. Balke, and L. Wolf, Eds. Bonn: Gesellschaft für Informatik e.V., 2012, pp. 843–852.

 ISBN: 978-1-61208-781-8

[12] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, “Seamless model-based development: From isolated tools to integrated model engineering environments,” Proceedings of the IEEE, vol. 98, no. 4, 2010, pp. 526–545.

[13] C. Reichmann, M. Kiihl, P. Graf, and K. Müller-Glaser, “Generalstore - a case-tool integration platform enabling model level coupling of heterogeneous designs for embedded electronic systems.” 06 2004, pp. 225 – 232.

[14] C. Reichmann, D. Gebauer, and K. D. Müller-Glaser, “Model level coupling of heterogeneous embedded systems,” in 2nd RTAS Workshop on Model-Driven Embedded Systems, 2004.

[15] B. Y.-L. Kuo, T. Hentrich, B. M. . Good, and M. D. Wilkinson, “Tag clouds for summarizing web search results,” in Proceedings of the 16th international conference on World Wide Web, C. Williamson, Ed. New York, NY: ACM, 2007, p. 1203.

[16] M. Schindler, A. Rausch, and O. Fox, “Clustering source code elements by semantic similarity using wikipedia,” in Proceedings of 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2015, pp. 13–18.

[17] M. Fowler, Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.

 ISBN: 978-1-61208-781-8