Sorokin, Radimir; Patil, Sandeep; Vyatkin, Valeriy

Novel development tool for IEC 61499 based on domain-specific languages

Published in:
IFAC-PapersOnLine

DOI:
10.1016/j.ifacol.2022.04.233

Published: 05/05/2022

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY-NC-ND

Please cite the original version:

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.
Novel development tool for IEC 61499 based on domain-specific languages

Radimir Sorokin * Sandeep Patil ** Valeriy Vyatkin ***
* Department of Computer Technologies, ITMO University, Russia (e-mail: rad.sorokin@gmail.com)
** Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden (e-mail: sandeep.patil@ltu.se)
*** Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland & Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden (e-mail: vyatkin@ieee.org)

Abstract:
This paper presents an example of a systematic approach based on domain-specific languages to implement Function Blocks Modelling Environment (FBME): an integrated development environment (IDE) for the international standard IEC 61499. The new open development environment for the standard of growing importance in the Industry 4.0 realm would contribute to the broader adoption of open standard-based development practices and facilitate efficient automation software development.

Keywords: IEC 61499, distributed control application, integrated development environment, domain-specific language, modular architecture

1. INTRODUCTION

Industrial programming environments are often based on the programming concepts and architectures defined by informal and semi-formal documents and somewhat formal descriptors. All these foundations undergo change over time, for example, as a result of international standardisation processes. Programming implementations of such concepts include integrated development environments (IDEs) with various graphical editors and visualisation aids. Implementing the changes introduced to the foundations in the supporting tools is a tedious and resource-consuming task. A systematic approach to reducing the effort is based on domain-specific languages (DSLs), bridging the gap between the foundations of the programming concept and its implementation in an interactive software tool. This paper presents an example of such an approach by implementing Function Blocks Modelling Environment (FBME): an IDE for the international standard IEC 61499.

The new open development environment for the standard of growing importance in the Industry 4.0 realm would contribute to the broader adoption of open standard-based development practices.

The development of a robust and modern IDE for this new automation engineering concept has many challenges. This paper attempts to address some of them in a constructive way.

The paper is structured as follows. In Section 2, the challenges of IDE creation for IEC 61499 are summarised. Section 3 compares different technologies for building IDEs and DSLs. Section 4 examines features of JetBrains MPS that assist in building the IDE and showcases how they are used in building the FBME. Section 5 gives a glance at the current architecture of FBME. Section 6 discusses other approaches to building an IDE and revises current solutions on IEC 61499 development. In Section 7, the further steps in the development of FBME and associated research are described.
• **Evolution of the standard.** With more use of IEC 61499 by developers, more ideas on how to engineer function block (FB) programs more efficiently occur. Some of such ideas systematise over time and then might crystallise into changes in the newer versions of the standard. For example, the concept of adapter connection has been extended in the second edition of the standard to allow plugs and sockets to be a part of a basic function block. Implementation of such changes (including user code migration features) should be easy for IDE authors.

• **Support for extensions of the standard.** Many research groups are inventing new extensions and deviations of the standard. For example, a dynamic adapter extension was proposed in Kajola et al. (2021). The ability to easily integrate such results in the form of an IDE plugin is essential to ease the adaptation of such extensions.

• **Composability with other languages and tools.** On the one hand, function blocks’ programs might embed different notations such as Structured Text (ST), Ladder Diagrams (LD), etc. In the future, more languages and tools might be supported to be composed with function blocks’ programs; for example, Bashev et al. (2020) propose PoST (Process-oriented Structured Text), an extension of the ST language that contains unique features of state-based programming which are missing in the existing languages for programmable logic controllers (PLCs). On the other hand, the entire concept of system development of the IEC 61499 standard is getting integrated with other related technologies, such as IEC 6131-3, AutomationML and Open Platform Communications Unified Architecture (OPC-UA) in such engineering frameworks, as the Open Process Automation Framework (OPAF) as proposed by Montague (2020).

• **Support for generative approaches.** In automation software engineering, there are generative approaches that rely on automatic generation of function blocks’ programs based on design documents formulated via higher-level domain-specific language.

• The transition from existing programming paradigms to the new paradigm of IEC 61499 will inevitably raise the issue of automated code migration of the existing projects such as Hussain and Frey (2005); Dai et al. (2013); Sunder et al. (2008).

Another global trend in software tools is their cloud-based implementation using a web interface. This architecture facilitates group work on projects by globally distributed teams of developers. It also enables the use of mobile devices for program development and testing. The cloud-based architecture also facilitates continuous integration and delivery. Besides, in the automation context, it can enable integration with cloud-based simulation facilities. Therefore, it can be anticipated that cloud-based tools would have great importance in the IEC 61499 domain. Building such a tool that covers all topics above from scratch is an overwhelming task. Fortunately, existing development environments, e.g. MBEDDR (Voelter et al. (2012)), FASTEN (Ratiu et al. (2019)), Agile Law Execution Factory (ALEF), etc.

The authors hypothesise that the DSL support by MPS would enable more efficient development of an open engineering environment for IEC 61499 (or for industrial automation software engineering) with the mentioned features than other existing tools or platforms.

This paper’s intended scientific contribution consists of developing methods for the creation of a “living IDE” that automatically follow changes in the supported programming languages.

### 3. OVERVIEW OF IDE/DSL CREATION TECHNOLOGIES

This section examines different technologies for building IDEs and DSLs and compares their features relevant to developing an IDE for IEC 61499 standard.

A common approach of building IDEs consists of basing upon existing IDE platforms, such as IntelliJ IDEA or Eclipse. As an alternative, language workbenches can be used for building an IDE on the top of a DSL designed with one of the language workbenches. XText (Efftinge and Völter (2006)), Spoofax (Kats and Visser (2010)) and JetBrains MPS language workbenches are considered for comparison. All of the compared language workbenches use either Eclipse or IntelliJ IDEA as a foundation and inherit their common features. There are also other modelling tools, such as GME (Generic Modeling Environment), which is used to model IEC 61499 entities in Zhou et al. (2016), and its newer version WebGME (Maróti et al. (2014)) are considered.

A comparison is presented on the following subjects, relevant to building a comprehensive IDE for IEC 61499 standard:

• **DSL modelling.** Language workbenches MPS, XText and Spoofax provide most of the valuable features such as the definition of a language structure, constraints, reference scoping and type-system. The set of features of GME tools and Eclipse Modelling Framework (EMF) is shortened to structure and constraints definition. There is no common option in the IntelliJ platform to model DSLs.

• **DSL composability and extensibility.** EMF, XText, Spoofax and MPS frameworks provide the support of modular language definition and provide features and case studies.

• **DSL evolution and versioning.** MPS provides dedicated support for model and language versioning and facilitates automatic model’s migration between
language versions. In other frameworks, such support usually has to be developed manually.

- **Support of generative approaches.** Most of the language workbenches and modelling tools provide support of generative approaches to produce executable code from a model, such as model-to-model transformations.
- **Support of projectional editing.** JetBrains MPS features an elaborate technology of projectional editing. Some scenarios of graphical visualisation and editing are also presented in GME, WebGME and EMF frameworks.
- **Guidelines for IDE creation.** IntelliJ IDEA and Eclipse platforms provide documentation on the topic of building an IDE. Developing IDE with XText and Spoofax follows the same guidelines as for Eclipse. MPS provides its tool-chain for building an IDE upon developed DSL.
- **GUI of modern IDE.** IntelliJ IDEA and Eclipse platforms have a graphical user interface (GUI) that has proven to be modern and functional by years of many users’ experience. The platforms also provide toolkits that can be used in a plugin to develop a canonical GUI for its functionality. The language workbenches based on these platforms also share these features.
- **Built-in plugin system.** IntelliJ IDEA and Eclipse platforms provide support for plug-in development and management. Language workbenches that are based on such platforms also inherit such functionality.
- **Cross-platform support.** IntelliJ IDEA and Eclipse platforms and language workbenches based on them can run on all standard operating systems such as Windows, Linux or macOS. GME is runnable only on Windows, while WebGME has a web-browser solution.

The summarised comparison is outlined in Table 1. It shows that JetBrains MPS has the most comprehensive set of features to build an IDE for IEC 61499.

### 4. IDE DEVELOPMENT WITH MPS

JetBrains MPS is a language workbench for building DSLs based on projectional editing. In addition to DSL creation, it also features instruments for shipping IDE support for developed DSLs along with tools for agile DSL development such as incremental migrations and language composability features.

This section examines and identifies MPS features that assist in building a comprehensive IDE for IEC 61499 standard with desired requirements. This makes a research foundation for the subsequent development.

#### 4.1 Declarative definition of DSLs

As the first step of DSL engineering in MPS, a user has to describe which entities are present in the domain and how they form the structure of DSL code. To achieve it, MPS provides language engineers with a set of own meta-DSLs that describe domain entities in a declarative way. Let us examine how it is possible to build some of the entities of the IEC 61499 standard using MPS.

As the standard prescribes, the specification of a function block in the network consists of its name, a name of its type, a set of actual parameters assigned to input variables and its visual position on the function block diagram (FBD). Such notion of function block can be modelled using the following concept declaration in Fig. 1 in MPS.

The concept declaration states that a node of this concept aggregates a ‘position’ node that describes its position on FBD and a list of ‘parameters’ nodes that specifies parameter assignments if any present. It also states that the nodes reference its type declaration represents with a distinct FBTNodeDeclaration concept. The concept declares no primitive properties except the ‘name’ property, which is derived via the inherited INamedConcept concept.

Given such declaration, a function block can be instantiated in the user code and operate programmatically with these instances using a meta-DSL. As a trivial example, it is possible to define a simple text-like editor for function block instances (Fig. 2) in a declarative way.

![Fig. 1. Declaration of function block concept.](image)

![Fig. 2. Declaration of function block editor.](image)
An example of an editor instance produced by shown declaration is presented in Fig. 3. Except for a textual representation, the constructed editor is shipped with smart editing support. For instance, the reference to the function block type has such useful features as an auto-completion, suggesting available function blocks, and navigation to the specified type declaration.

![Fig. 3. Example of function block editor instance.](image)

**4.2 Extensibility of MPS with Java extensions**

Due to domain specifics, different languages modelled in MPS might require additional tools to be developed and embedded into MPS. Therefore MPS is designed to be extensible in various ways.

Efficient design support of the IEC 61499 standard requires highly elaborated diagram editors to operate with FBDs. Even though MPS provides a projectional editing framework, such complex diagram editors are not available out of the box. Fortunately, the existing projectional editor of MPS can be extended with Java code to support the required editor for block diagrams of IEC 61499.

Conceptually, MPS constructs a cell tree to implement an editor for an arbitrary set of visible objects. Each cell is associated with a specific region of an editor’s canvas and defines how this region is rendered and what particular actions (completion, context menu actions, keystroke bindings) users can apply within it. Programmatically, a cell is represented with the `EditorCell` Java interface, and MPS provides a set of predefined implementations. If the language designer requires some specific behaviour by the editor, he/she can add another suitable implementation of the cell interface and integrate it in the cell tree constructing process.

A custom cell class has been implemented to support customisable diagram editors. With the use of it, diagram editors have been implemented of ECC and FBS, as shown in Fig. 4.

![Fig. 4. Example of FBD (top) and ECC (bottom) editors.](image)

**4.3 Multiple projections of a model**

MPS utilises model-view-controller pattern to make DSL editors. In particular, a user code is presented as an in-memory model. The model’s editor (controller) renders it in the form of a cell tree (view) on the screen and enables editing a model via user interactions.

This approach enables an ability to present a user code via several views (or projections, in MPS terms). For instance, an FBD can be presented to a user in textual notation and as a diagram. The architecture of a multi-projectional editor that presents how a user interacts with a document via several projections is presented in Fig. 5.

![Fig. 5. Architecture of multi-projectional editor.](image)

**4.4 IDE support and plugin system**

MPS is based on the IDEA Platform and inherits a significant piece of its features like native UX (user experience), file-system management, plugins management, etc. IDEs based on MPS also get this functionality out of the box.

The plugin support is a critical feature for all modern IDEs. This part of the IntelliJ IDEA platform is also presented in FBME. This feature enables building custom extensions to FBME. For example, Tereshchuk (2020) developed an FBME plugin that performs formal verification of IEC 61499 applications employing model-checking with NuSMV and explaining counterexamples resulted from the model-checking.
FBME is designed to be modular and extensible. The development of a modular architecture includes dividing the developed functionality into distinct components and describing the components of each module. All modules fall into three categories:

- **The FBME platform module.** This module, along with its dependencies, forms the core functionality of an IDE.
- **Platform extensions.** These modules provide additional functionality to the IDE.
- **Primitive modules.** These modules consist of utilities or integrated external tools and libraries. Modules of this type are used and shared by the platform and its extensions and implement their features.

The set of platform extensions and primitive modules is open and can be extended by third-party solutions and integrated into the FBME via plugins.

At this point, FBME consists of six modules. A graphical representation of these modules and their dependencies are shown in Fig. 6. A brief description of the content of these modules is as follows:

1. **JetBrains MPS** module consists of software and user interfaces provided by the JetBrains MPS platform.
2. **FBME Core library** provides an API for working with standard systems using Java programmatic interfaces.
3. **FBME Platform** provides the basic functionality of the IEC 61499 systems’ development environment, namely, creating and editing the systems, import/export of the systems, launching applications on available devices. It also provides a set of extension points for connecting to running devices and providing custom editors for system declarations.
4. **Rich editing** expands the basic editing capabilities of IEC 61499 systems by adding diagram editors for functional block diagrams and ECC diagrams. It also provides an API for embedding visual debug information into the editor.
5. **MPS Scenes** implements a set of components to support visual editing in the JetBrains MPS environment. This module was developed to simplify the creation of functional block diagram editors.
6. **4diac FORTE integration** implements support for connection to devices of the 4diac FORTE runtime environment and local launch of these devices. The module also exports a library of function blocks supported by 4diac FORTE.

6. RELATED WORK

The need to build comprehensive engineering tools appears in many different domains nowadays. There are several approaches to tackle this task. The naive approach is building the whole engineering tool from scratch. Unfortunately, it is a very resource-consuming task – building and maintaining such applications seems to be only feasible for commercial products, e.g., IntelliJ IDEA, Eclipse, Visual Studio.

A more feasible approach is to reuse the existing open platform for building engineering tools and extend it with domain-specific features. The main benefit of basing on the top of the existing platform is that development is focused on delivering features unique to a distinct domain rather than on repetition of functionally to present in all other tools.

Language workbenches ease building an engineering tool even more by providing tools to wrap the domain in a language, declaratively create the language and integrate it in an IDE. There are many language workbenches present, and all of them have their pros and cons, which are evaluated in Erdweg et al. (2015). The MPS platform is chosen to build FBME because it has projectional editing features, language composition, extensibility, etc.

Generative approach is influencing the development of programming tools. For example, design documents describing the physical layout of energy distribution networks were used in Yang et al. (2019) for generation of control applications in IEC 61499.

The most notable open-source development of an IEC 61499 compliant IDE is the Eclipse 4DIAC project. To the best of the authors understanding, the Eclipse framework, despite all its power, lacks the DSL support means comparable with that of MPS. Nevertheless, Eclipse 4DIAC is seen as a reference case for our development.

Integrated development environments tend to include many functions supporting the entire life-cycle of software development, testing, commissioning and maintenance. For example, related research works propose novel verification and validation methods for IEC 61499 based on model checking, e.g., Drozdov et al. (2021). It can be anticipated that integration of such features into the IDEs would be in demand.

7. FUTURE WORK

The future work plans go along the following directions. Firstly, the MPS feature of language compositability will be employed to deliver controllable IEC 61499 standard extensions. In particular, the FBME infrastructure will be elaborated so that such extensions as dynamic ports adapters or PoST language integration will be possible.

Secondly, the MPS feature of language versioning and migrations will be utilized to handle future possible IEC 61499 standard updates. In particular, migration scripts might be used to tackle a possible format evolution of IEC 61499 standard documents.

Thirdly, model-to-model transformations featured in MPS might be adopted to design a high-level DSL of defining...
cyber-physical models from which IEC 61499 models are generated automatically.

Once the FBME development achieves the first usability milestone, an experiment with university student groups to compare its efficiency with other tools will be set. At last, currently, several approaches are being developed to enable Web solutions for MPS, such as modelix\(^1\) and WebEditKit\(^2\). Integration with such frameworks will be considered to enable Web and collaborative development of IEC 61499 systems.

8. CONCLUSIONS

The paper outlines the hypothesis that the use of language workbenches has to increase the efficiency of developing open engineering environments for industrial automation software engineering and attempts to demonstrate its validity constructively by presenting the development process of an IDE for IEC 61499 with the use of JetBrains MPS language workbench. The research has shown that highlighted features of JetBrains MPS help make the IDE’s architecture modular and build an IDE efficiently in the early stages of development. The outgoing development of FBME that elaborate the IDE for the production use will substantially uncover the validity of the proposed hypothesis, which will be addressed in future work.

ACKNOWLEDGEMENTS

This work was supported, in part, by the HORIZON2020 project 1-SWARM co-funded by the European Commission (grant agreement: 871743).

The authors are grateful to the MPS team at JetBrains for building MPS and their support in developing FBME.

REFERENCES


