IEC 61499 Function Block Implementation of Intelligent Mechatronic Component

Cheng Pang, *Member IEEE*, Valeriy Vyatkin, *Senior Member IEEE*
The University of Auckland
cpan024@aucklanduni.ac.nz, v.yatkin@auckland.ac.nz

***Abstract*-This paper presents a new approach of implementing Intelligent Mechatronic Component (IMC) using the IEC 61499 Function Block standard. In particular, this paper tries to address two issues during the engineering process of IMCs: a systematic approach of developing and organizing IMC elements and a seamless and scalable way supporting IMC composition and reconfiguration. A case study implementation example is presented to illustrate how the results from this work can be applied and integrated in Function Block development tools.**

I. INTRODUCTION

Since the release of the IEC 61499 Function Block (FB) standard [1], a number of studies have been conducted to investigate its practical applications in automation systems. Although the industry has been aware of the benefits brought by IEC 61499, the new standard has still not yet been widely adopted. One of the main reasons is the immaturity of the development tools, especially the lack of sophisticated IDE, and the well-integrated design methodologies facilitating component-based design throughout the entire design cycle of automation systems.

One of the IEC 61499 features is support of interoperability among devices from different vendors. The standard provides an open XML format for storing code modules and exchanging them between tools and devices from different vendors. However, the standard is not intended to cover all aspects of the automation engineering process, such as visualization, simulation, verification, and deployment. As a result, vendors of IEC 61499 IDE and tool may choose to incorporate their own design technologies or apply third-party approaches to comprehend the design methodology covering the entire design cycle of their products and therefore introduce their new, either open or proprietary, storage formats. These formats usually cannot be recognized by other tools hence violating the original intention of the IEC 61499 standard. For example, the nxtControl IDE [2] introduces the concept of *Computer Automation Types*, which contains control component, visualization, documentation, and process connection in a single unit saved in their own proprietary *IEC 61499 Solution* format. Similarly, ISaGRAF [3] uses encrypted XML format for storing IEC 61499 applications. Contrarily, instead of providing a complete solution, some IEC 61499 design tools only focus on certain aspects of the design process, typically the composition of control logics, and often use other existing tools as plug-ins to provide common functions or themselves are parts of larger domain-specific design frameworks. As a result, these tools save the designs as standard IEC 61499 XML files and provide translators to interface the external tools, which make the FB designs exchangeable and resuable but also impose the difficulties to adapt existing design methodologies on various tools to provide comprehensive design solutions.

The involvement of different tools and technologies in the engineering process of control systems requires a consistent data exchange mechanism to ensure smooth workflow during the whole project. The inability to exchange data across the heterogeneous tools seamlessly while maintaining the integrity of semantics will incur extra difficulties, time, and consequently costs on the development. This paper presents the first step to seamlessly incorporate IEC 61499 standard with other tools and standards to support the development lifecycle of control program for automation system following a systematic design methodology. The solution consists of:

- an engineering methodology based on the concept of Intelligent Mechatronic Component (IMC) [4] to support the design lifecycle of automation systems; and,
- a corresponding generic data exchange mechanism based on IEC 62424 standard [5] as the key for the integration of heterogeneous tools in the FB development, so that each tool can work on its native file format simultaneously.

This paper is structured as follows. Section II introduces the general concepts of IMC. Section III outlines the overall modeling methodology of IMC using IEC 61499 FBs. Then, Section IV elaborates the generic data exchange mechanism defined by IEC 62424 with its semantic extensions to support FB implementation of IMC in Section V. A concrete example illustrating the development of FB IMCs is also included here. Finally, this paper is concluded with the contribution of this work and future research perspectives.

II. INTELLIGENT MECHATRONIC COMPONENT

The concepts of Intelligent Mechatronic Component (IMC) was first introduced in [4] to the automation software design domain and then further discussed and extended in [6-8]. The general idea of IMC is that, machines or mechatronic components come with pre-programmed software, including plant models, control programs, and any necessary network interfaces. Each IMC can contain the following items:

- *Mechatronic component*: a physical functional device with sensors, actuators, and electronic circuits;

- *Embedded control device*: computing devices with interfaces to the sensors, actors, and networks; and,
- *Software components*: a set of data and control logics implementing various automation functions. These elements provide the IMC's autonomy and cooperation.

New machines or automation systems are constructed as a result of integrating these elements of existing IMCs. Fig. 1 below illustrates the overall process of building new systems using IMCs.

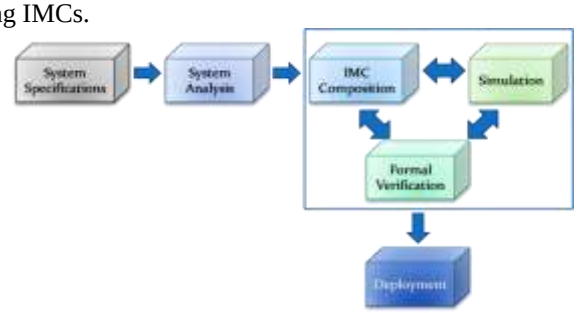

Fig. 1. IMC Engineering Methodology

The process starts with capturing the requirements of the new system. According to the system specifications, analysis is performed to determine the actual implementation details, such as the design layout, communication protocols, and so on. The system's overall structure will then be conceptualized using computer-aided design tool. In such tool, the simulation models, including plant and control models, of IMCs will be picked from the IMC repository provided by their vendors and placed into the design space for closed-loop simulation first. If the simulation results prove that the predefined testing scenarios are correct, corresponding formal model can be generated to perform a more exhaustive system validation. Once the verification results are satisfactory the IMCs' control programs can be deployed to the physical system.

This unified hierarchical architecture greatly facilitates the engineering and maintenance processes of current automation systems. However, to achieve this vision, a proper software architecture as a form of knowledge organization of IMC software components and a mechanism supporting automatic integration and deployment of IMCs are substantial. In this research, the IEC 61499 Function Block standard is adopted to handle the control organization, plant model development, and simulation of IMCs following the modeling approach we previously developed in [9]. The IEC 62424 standard, on the other hand, is applied to structure and categorize the IMCs in the design process and manage the later deployment step.

Next section will first clarify the reasons of using the IEC 61499 standard to develop IMC models and then the FB modeling methodology for IMC is introduced.

III. Systematic IMC Modeling Methodology

One important feature of IMC is the ability to encapsulate the component's functionality into reusable portable software units. The design of new machines or production systems' behavior and functions can be reduced to the selection, scheduling, and composition of such units in a plug-and-play way. However, to implement the plug-and-play integration of IMCs from different developers and vendors a unified or standardized software architecture, network interfaces, and data formats must be agreed and applied. The new IEC 61499 standard established such framework for agile development of component-based automation systems, which are portable, reconfigurable, and interoperable among different vendors. More importantly, the standard provides a single mechanism to represent and encapsulate both plant and control models using the same language and architecture, which perfectly suits the concepts of IMC.

Given the standardized development means provided by IEC 61499, the software repository for each IMC must be also organized in a uniform way and moreover a mechanism for seamless integration of IMCs must be invented. The interface-based composite Model-View-Control (MVC) design pattern we previously developed [9] can play such a role. Fig. 2 (a) shows the schematic diagram of composing the software components of two IMCs each of which consists of a Model FB, a View FB, a Control FB, and peripheral HMI FBs. The interactions between these functional aspects, such as the Model-View interaction, Model-Model interaction, Model-Control interaction, and Control-Control interaction, are handled by a uniform mechanism. The resultant new IMC again follows the same MVC topology as shown in Fig. 2 (b).

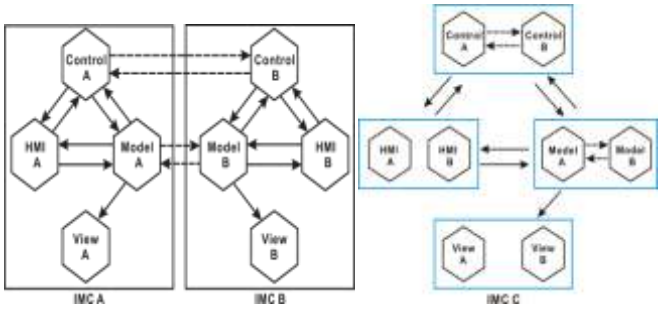

Fig. 2. Composite Model-View-Control IMC Design Pattern: (a) Composing two IMCs, and (b) Resultant IMC

This composite MVC design approach provides the means for designers to manually integrate the FB models of IMCs. A more important step towards IMC concept is to have software tools support automatic integrations of these FB models and their deployment to the physical devices. The IEC 61499 standard does not provide such facility as it is not within the standard's scope. As a result, extra information must be provided for the tools to layout, integrate, and deploy these IMC FBs. Next section introduces the IEC 62424 standard and the generic data exchange mechanism we borrowed from it to specify IMC FBs' integrating points, layout information, deployment polices, and so on.

IV. IEC 62424 Standard and CAEX

The IEC 62424 standard was published in 2008 with the original aim to specify the representation of process control engineering requests in Process & Instrumentation Diagrams (P&ID) along with the data exchange mechanism between P&ID tools and Process Control Engineering and Computer Aided Engineering tools. To implement a generic data

exchange mechanism, an abstract object-oriented data format called Computer Aided Engineering Exchange (CAEX) was developed. Although CAEX is originally designed for the engineering in process industries, it has been adopted by AutomationML [10] as the technical basement to achieve seamless data storage and exchange covering all aspects of automation engineering activities from plant planning and functional engineering to the final commissioning. Indeed, CAEX can be applied to all types of static object information, such as various design topologies, as well as for non-technical applications.

A. General CAEX Concepts

The essence of CAEX is the XML-based CAEX schema, which unites model and meta-model techniques into a single template. As a meta-model, CAEX provides a syntactic but flexible means for defining specific semantics and structure on the stored object information. On the other hand, CAEX standardizes an object-oriented way of storing information to establish a generic data exchanging mechanism. Therefore, a CAEX file created in a specific application can also be recognized by other applications supporting CAEX model but with their individual interpretation of the stored information. This decoupling mechanism preserves the data integrity while allowing heterogeneous manipulation on the data objects.

Fig. 3 below illustrates the top-level view of the CAEX schema which describes the valid structure of a CAEX file and the relationships between the meta-elements.

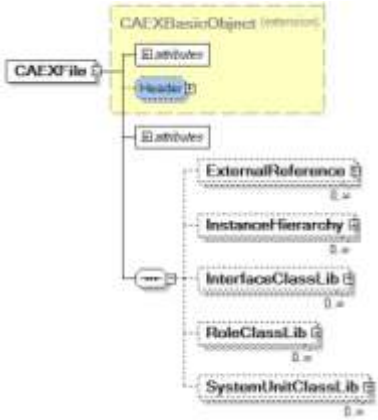

Fig. 3. Top-Level View of CAEX Schema

The root element of CAEX schema is called *CAEXFile* which contains the definitions of three class libraries:

- *SystemUnitClassLib* is a library lists arbitrary number of objects of type *SystemUnitClass*, which can be used to describe, for example, physical or logical plant objects or units including their specifications, internal structure, and operating parameters. Each *SystemUnitClass* type consists of attributes, interfaces, and recursively nested and interrelated *InternalElements* describing the internal structure of the modeled system. Within the library, *SystemUnitClass* types are organized as a tree to depict the user's library hierarchy and the classes' relationships.
- *RoleClassLib* also collects objects of type *RoleClass* as a tree. Each *RoleClass* type only consists of attributes and interfaces as its purpose is to describe the abstract requirements of objects.
- *InterfaceClassLib* again arranges *InterfaceClass* types as a tree. *InterfaceClasses* are used to define interfaces and relations for *RoleClasses* and *SystemUnitClass* types.

The class types listed above can be instantiated and stored in *InstanceHierarchy* to structure the system being modeled. By using a reference mechanism and hierarchical structures, CAEX supports the concepts of inheritance and composition on class types and instances. At last, the *ExternalReference* element defines the mechanism to reference external data and interlink external object information to local objects.

B. Basic CAEX Modeling Example

Without supplying further semantic definition, one typical usage of CAEX is to model the structure of static object information, such as topologies of physical plants, documents, or even Petri nets. The following example demonstrates how a CAEX model is developed to describe the structure of a mechatronic device called FESTO Distributing Station (DS) from the early planning phrase to final implementation. As depicted in Fig. 4, DS consists of two parts: a Stack Magazine (SM) module and a Changer module. The main function of DS is to transfer the workpieces inside the magazine barrel of SM to the downstream station via the suction cup of Changer.

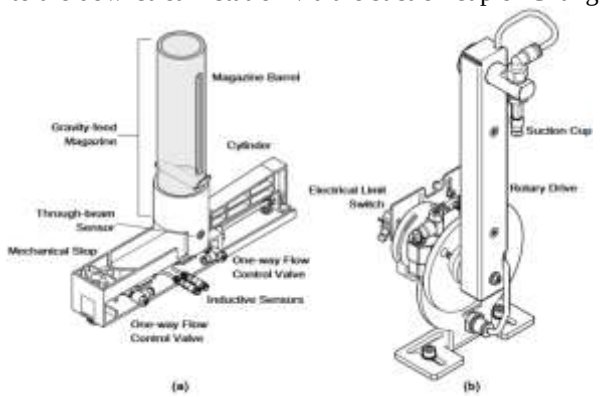

Fig. 4. (a) Stack Magazine and (b) Changer Modules [11]

Depending on the required modeling granularity, the object structure of DS can be stored up to a certain level of details. This example only considers the composition layout and object relations in the hierarchical structure of DS to present the basic usages of CAEX elements and concepts that will be later used to define further semantics for describing IMC FBs.

The modeling process starts with identifying components in DS. Each component is treated as a role outlining its abstract functions and properties as depicted in Fig. 5.

Fig. 5. RoleClassLib for Distributing Station

The properties and requirements of each role are stored as attributes inside the corresponding *RoleClass* element. For

instance, the *InductiveSensorRole* element specifies the operating temperature, physical size, and sensing range of the inductive sensor that the SM module requires. According to the requirements, a concrete inductive sensor with the correct dimensions, suitable temperature tolerance, and sensitivity is selected from the *SystemUnitClassLib* library, which stores and categorizes the detail information of, for example, the standard FESTO components and sensors as shown in Fig. 6.

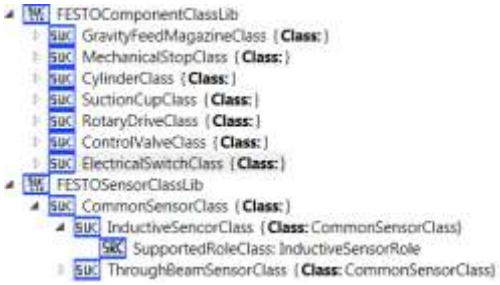

Fig. 6. SystemUnitClassLib for FESTO Components

Each *SystemUniClass* element can support multiple roles as indicated by its child *SupportedRoleClass* elements and the inheritance relations across *SystemUnitClass* elements can be identified by the corresponding *Class* attribute. For example, the *InductiveSensorClass* element inherits all the properties from the *CommonSensorClass* with extra functions to support the *InductiveSensorRole* required by the SM module. The *SupportedRoleClass* element and the *Class* attribute can be used by tools to automatically identify and instantiate the suitable class for specific role during the IMC design process.

The classes selected from the *SystemUnitClassLib* will then be instantiated as internal elements and then assebmled inside *InstanceHierarchy* to define the internal structure of DS. As shown in Fig. 7, the *StackMagazineModule* element contains all the components used to build the SM module. The details and role requirements of the components are provided by the *SystemUnitClass* and *RoleClass* elements specified by the respective *Class* and *Role* attributes while extra properties, such as the position of the inductive sensor in SM can be specified as additional attributes under *InductiveSensor*.

Fig. 7. InstanceHierarchy for Distributing Station

At last, the connection between the components can be realized by defining their external interfaces. For instance, Fig. 8 shows the *WorkpieceInterface* used to describe the connections between the SM and Changer modules with the details such as workpiece flow direction, connection port, and possible workpiece type. This interface concept provides the mechanism to define the semantics for automatic connection of the IMC models.

Fig. 8. InterfaceClassLib for Distributing Station

V. Semantic Extensions and Demonstration Example

CAEX provides a standardized storage means of data in a syntactic way. The data templates defined by the designers provide the corresponding semantic specifications for tools to interpret the stored data. This section elaborates the data templates developed in this work to store the details of IMC FBs and the imposed semantics on the data structure for tools to interpret and manipulate the underneath FB models and hence to achieve automatic configuration and generation of new system designs.

In our template, the *RoleClassLib* stores a number of roles containing abstract properties and requirements that can be associated with *InternalElement* or *SupportedRoleClass*. For the first case, the *RoleClass* delivers common attributes and interfaces to the associated *InternalElements* while allowing concrete requirements to be specified in the *RoleRequirement* elements. For the second case, the *RoleClass* tags indicate what roles the current object can play. By examining the *SupportedRoleClasses*, software tools can extract necessary information from the role's attributes and interfaces to, for example, automatically configure the object and hence set up suitable deployment environment. The following Fig. 9 lists the *RoleClassLib* templates developed in this work.

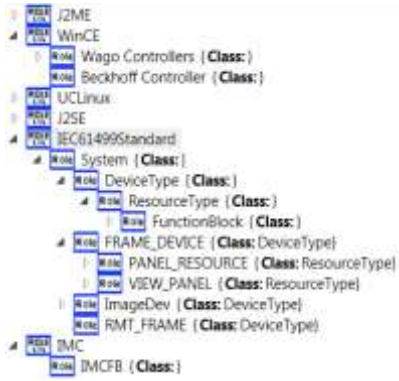

Fig. 9. RoleClassLib Templates

The most essential *RoleClassLib* is the *IEC61499Standard* library, which contains *RoleClass* representations of various IEC 61499 entities from the generic *System*, *DeviceType*, and *ResourceType* to the concrete inherited *FRAME_DEVICE* and *PANEL_RESOURCE* types. These concrete *RoleClasses* store the required parameters and configuration information as attributes and interfaces whose actual values will be specified when the roles are associated. The *IEC61499Standard* library is typically used to define IEC 61499 applications' topology and therefore provide the information of possible deployment configurations. Moreover, the hierarchy structure implies the compatibility of the resources and devices. For example, the

VIEW_PANEL resource must only be deployed to the *FRAME_DEVICE* not any the other devices.

The implementation details of IMC FBs, especially the configuration of the MVC components and the connection semantics of their signal interfaces, are stored as instances of *SystemUnitClass* under the corresponding *SysemUnitClassLib*. As FBs can easily have tens of I/O ports, direct connections of these signals can significantly crowd the original design and worsen the readability of the resultant *SystemUnitClass*. Therefore, in this work, the IMC FB models are developed following the composite MVC design pattern as described in Section III, which uses IEC 61499 *Adapter Interfaces* (AIs) to logically group related signals into a single link. Similar to the concept of software interfaces, AIs are used to identify the compatibility of signals as only matched AI types can be connected. Fig. 10 shows the SM IMC's Model, View, and Control FBs whose AIs are connected following the same topology as illustrated in Fig. 2.

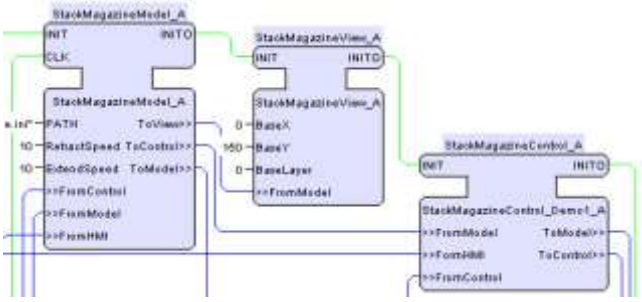

Fig. 10. Model, View, and Control FBs of Stack Magazine

Referring to the CAEX model, the SM IMC is modeled by the *SystemUniCLass StackMagazineIMCFB* with all MVC FBs represented as interlinked *InternalElements* inside it as shown in Fig. 11 below.

Fig. 11. SystemUniClassLib Templates

The *StackMagazineIMCFB* inherits the *GenericIMCFB* class indicates it represents an IMC FB which has attributes: *IMCFolderReference* and *IMCPreviewPath* informing the location and preview path of the IMC model. The CAEX models of MVC FBs are identified by the associated *IMCFB RoleClass*, which has a requirement, *IEC61499FBReference*, specifying the path to the FB file. The *SupportedRoleClasses* of, for example, *ModelFB* indicate that it can be deployed to the *ImageResource* and *PANEL_RESOURCE*. Similarly, the *DS_HMI_Set_Demo1* collects normal FBs and hence inherits the *GenericFunctionBlock* class and associates with role *FunctionBlock*. On the other hand, the IO interfaces of the FBs are represented as *ExternalInterfaces* of types from *IEC61499InterfaceClassLib* as listed in Fig. 12.

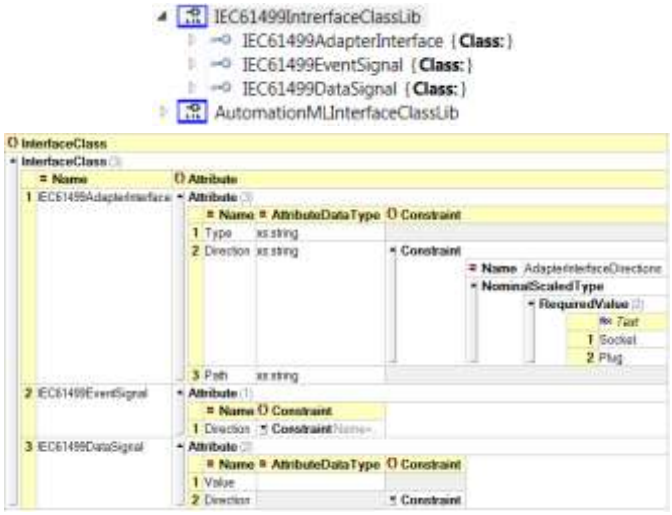

Fig. 12. InterfaceClassLib Templates

The *IEC61499AdapterInterface* models AI port, which has attributes *Type*, *Direction*, and *Path* specifying the AI type, port direction (socket or plug), and the path to the AI file. On the other hand, the *IEC61499EventSignal* interface represents IEC 61499 event port with attribute *Direction* specifying whether the port is input or output. Similarly, in additional to the *Direction* attribute, the *IEC61499DataSignal* interface has a *Value* attribute defining the type and value for the data port.

Finally, the connections between the MVC FBs are stored as *InternalLinks* of the parent *SystemUnitClass* which enables automatic configuration of the MVC FBs when the parent class is instantiated. Each *InternalLink* has three attributes as exemplified in Fig. 13 where attribute *Name* defines the link's name and attributes *RefPartnerSideA* and *RefPartnerSideB* specify the link's source and destination port using the format *FB_UniqueID:IO_Name*. For instance, the entry {912a71b9-cb28-4af1-8da7-cec6dfa44547}: INITO refers to the unique ID of StackMagazineIMCFB's *ModelFB* element and the port name is INITO. This unique ID reference mechanism is used to avoid conflict when cross referencing objects especially in the component-based design environment of IMC FB.

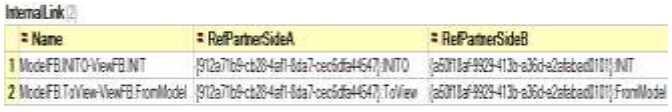

Fig. 13. AI Connections as InternalLink

Existing IMC FBs from the *SystemUnitClassLib* can be instantiated as *InternalElements* and then placed, connected, and configured inside *InstanceHierarchy* to further define the

structure of new IMC FBs, which can be then stored as new *SystemUnitClass* back to the library for later use. During the design process, new elements such as *InternalLinks*, attributes, and *InternalElements* can be added to detail the new system's requirements following the same mechanism as designing *SystemUnitClasses*. Fig. 14 exemplifies an *InstanceHierarchy* representing the structure of the DS module as a result of configuring the existing SM and Changer models with HMIs.

Fig. 14. InstanceHierarchy Template Example

The DS system consists of two devices, *DISPLAY* and *HMI* of type *ImagDev* and *FRAME_DEVICE* respectively. The MVC FBs of SM and Changer IMCs are deployed to *ImageResource Res1* under *DISPLAY* while the HMI FBs are installed in the *PANEL_RESOURCE* inside *HMI*. All the properties, interfaces, and internal connections between the MVC FBs are directly instantiated from the corresponding *SystemUnitClasses*. Further connections among the SM and Changer IMCs and other FB models are also specified as *InternalLinks* under elements *DISPLAY-Res1* and *HMI-Res1* respectively. According to this DS *InstanceHierarchy*, tools can automatically generate a new system configuration and determine suitable deployment polices based on the hierarchy structure and the supported roles.

At last, the CAEX reference mechanism is used to support modular development of IMC FBs. In particular, as shown in Fig. 15 the SM IMC model can be stored as external file and then referenced use the *ExternalReference* element where the *Path* attribute specifies the file location and *Alias* defines the actual tag name to be referenced in the new design.

Fig. 15. InstanceHierarchy Template Example

VI. Conclusions

The IEC 61499 standard established a unified framework for developing the logic part of next generation automation systems. However, the engineering process of modern control systems involves not only the control algorithms but also the visualization, simulation, and final deployment of the entire system, for example. These engineering aspects are not covered in the standard but must be supported by the IEC 61499 development tools. This paper presented an approach of implementing IMCs to bridge various engineering aspects of developing automation systems using IEC 61499 FBs with the generic data exchange mechanism CAEX to seamlessly integrate the heterogeneous tools involved in the entire design process. The demonstrative FESTO Distributing Station example illustrates the design flexibility using IMCs. The future research will focus on applying the CAEX templates to incorporate more practical engineering tools and standards.

References

[1] International Electrotechnical Commission, Function blocks for industrial-process measurement and control systems - Part 1: Architecture. Geneva: International Electrotechnical Commission, 2005.

[2] nxtControl. (2009, December). [Online]. Available: www.nxtcontrol.com

[3] ISaGRAF. (2009, December). [Online]. Available: www.isagraf.com

[4] V. Vyatkin, "Intelligent mechatronic components: control system engineering using an open distributed architecture," in Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference, 2003, pp. 277-284 vol.2.

[5] International Electrotechnical Commission, IEC 62424: Specification for representation of process control engineering requests in P&I Diagrams and for data exchange between P&ID tools and PCE_CAE, 1 ed.: International Electrotechnical Commission, 2008.

[6] C. Sünder, et al., "Functional structure-based modeling of automation systems," International Journal of Manufacturing Research, vol. 1, pp. 405-420, 2006.

[7] S.-M. Lee, et al., "A Component-based Distributed Control System for Assembly Automation," in 2nd IEEE Conference on Industrial Informatics (INDIN 2004), Berlin, Germany, 2004, pp. 33-38.

[8] K. Thramboulidis, "Model-integrated mechatronics - toward a new paradigm in the development of manufacturing systems," Industrial Informatics, IEEE Transactions on, vol. 1, pp. 54-61, 2005.

[9] C. Pang and D. V. Vyatkin, "Systematic Closed-Loop Modelling in IEC 61499 Function Blocks: A Case Study," in Information Control Problems in Manufacturing, 2009. INCOM09. 13th IFAC Symposium on, Moscow, Russia, 2009, pp. 199-204.

[10] L. Hundt, et al., "Seamless Automation Engineering with AutomationML," in 14th International Conference on Concurrent Enterprising (ICE 2008), Lisboa, Portugal, 2008, pp. 685-692.

[11] F. Ebel, et al., FESTO Distributing Station Manual. Denkendorf, 2006.