Intelligent Component-Based Automation of Baggage Handling Systems With IEC 61499

Geoff Black and Valeriy Vyatkin, *Senior Member, IEEE*

***Abstract*—Airport baggage handling is a field of automation systems that is currently dependent on centralized control systems and conventional automation programming techniques. In this and other areas of manufacturing and materials handling, these legacy automation technologies are increasingly limiting for the growing demand for systems that are reconfigurable, fault tolerant, and easy to maintain. IEC 61499 Function Blocks is an emerging architectural framework for the design of distributed industrial automation systems and their reusable components. A number of architectures have been suggested for multiagent and holonic control systems that incorporate function blocks. This paper presents a multiagent control approach for a baggage handling system (BHS) using IEC 61499 Function Blocks. In particular, it focuses on demonstrating a decentralized control system that is scalable, reconfigurable, and fault tolerant. The design follows the automation object approach, and produces a function block component representing a single section of conveyor. In accordance with holonic principles, this component is autonomous and collaborative, such that the structure and the behavior of a BHS can be entirely defined by the interconnection of these components within the function block design environment. Simulation is used to demonstrate the effectiveness of the agent-based control system and a utility is presented for real-time viewing of these systems. Tests on a physical conveyor test system demonstrated deployment to embedded control hardware.**

***Note to Practitioners*—It is well recognized that flexibility and reconfigurability of manufacturing systems pose a major challenge to their automation. There is also a belief among automation researchers that decentralized intelligent control will contribute significantly to the flexibility of manufacturing systems. Achievements in practical applications using decentralized control have been limited by the standing tradition of automation design with centralized logic executed on programmable logic controllers (PLCs). The new programming architecture defined by IEC 61499 is specifically designed for distributed applications and invalidates many existing assumptions about the difficulty of implementing these systems using partially or fully decentralized execution.**

In this paper, we present results of developing a distributed automation architecture based on IEC 61499 for airport baggage handling systems. Such systems are constantly undergoing reconfiguration, repair and expansion so flexibility is a vital design metric. We demonstrate several sides of "flexibility" enabled by IEC 61499, including component reuse, maintenance and human interaction, and reconfiguration.

***Index Terms*—Distributed factory automation, holonic control, IEC 61499, material handling systems.**

I. Introduction

MATERIAL handling is a field of automated systems that deals with movement of materials rather than of processing. Airport baggage handling systems (BHSs) is a representative example of such systems, known to everybody. Many of the issues faced in BHS are relevant to industrial automation in general and *vice versa*. In particular, of high importance are the goals of serving "rapidly changing markets" by "shorter time to market" and "increased customization," which have been cited for years in the industrial automation context.

From this perspective, conveyor-based BHS are considered desirable [1] partly because they are readily modified or reused to allow reconfigurable applications. The conveyor, therefore, seems a very good example of an application where an easily reconfigurable, intelligent mechatronic module might have considerable benefit. Moreover, flexibility in this context may refer both to physical reconfigurability or to the design process. The former is important due to the fact (also noted in [1]) that most airports exist in a permanent state of expansion and upgrade. The design flexibility would allow reuse of previously developed solutions which can help create new BHS faster and with higher quality assurance.

Despite these widely agreed needs, there has been arguably a poor level of achievement in reaching these stated aims. Current controllers for BHSs are based on conventional industrial control hardware and programming techniques. This includes a heavy reliance on programmable logic controllers (PLCs) for the low-level manipulation of actuators based on sensor data. However, the PLC-based centralized approach to control cannot be appropriately applied to all circumstances with some applications being too dispersed physically or demanding of processing to allow control from a single execution point [2]. The software used in PLCs is generally monolithic, increasing the difficulty of modification and maintenance and reducing scalability. Although with modern tools, the PLC code may be quite modular, there is still a considerable amount of effort required in order to reconfigure PLC code for a new BHS.

IEC 61499 standard [3] provides an architectural framework for the design of distributed and embedded control systems. It aims to become a direct successor to the current suite of programming languages for automation systems, allowing the development of applications running on multiple decentralized control platforms. IEC 61499 also promises to accommodate

Manuscript received June 30, 2008. First published January 09, 2009; current version published April 07, 2010. This paper was recommended for publication by Associate Editor J. Li and Editor V. Kumar upon evaluation of the reviewers' comments. This work was supported in part by the TechNZ and Glidepath Ltd. TIF grant and by the University Grant SRF 3607893.

G. Black was with the University of Auckland, Auckland 1142, New Zealand. He is now with Wellington Drive Technologies, Auckland, New Zealand (e-mail: Geoff.Black@wdtl.com).

V. Vyatkin is with the Department of Electrical and Computer Engineering, University of Auckland, Auckland 1142, New Zealand (e-mail: v.vyatkin@auckland.ac.nz).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2008.2007216

better the intelligent automation ideas emerging from many mature long-running projects. Amongst the various efforts, the need for a more distributed approach to control is generally accepted by [2] and [4].

Three approaches to distributed control worth mentioning. Early attempts at distributed control involved splitting a large application in smaller components and joining them together with communications to achieve the overall required behavior.

Multiagent control systems [5] take a different approach. Instead of creating applications by gluing together subprograms with communications to form a static distributed system, agents are designed to be autonomous actors in an environment where they perform local actions, while actively collaborating with other agents to achieve global goals [6].

A particular kind of agent-based systems are *holonic systems*. *Holonics* is a concept derived from observations of natural systems that consist of hierarchies of entities that may each be considered complete systems [7]. A holonic system is said to be made of "holons" which may be considered both as an entity in their own right, and as a component of a larger hierarchy known as a "holoarchy." While multiagent systems emerged largely from research in distributed artificial intelligence [8], the field of holonics was initially inspired by Arthur Koestler's "The Ghost in the Machine," which utilizes the concept in discussing evolutionary psychology, including the invention of the word "holon."

In the field of automation and manufacturing systems, holonics seems to be attractive because of the connotations of resilience to disturbance and adaptation in response to component failure that characterize many of the natural systems from which the holonic principle originates.

Early studies on the use of multiagent and holonic approaches in automation have shown that the most critical for their success are modularity and redundancy of the machinery. Therefore, material handling systems such as BHS can be considered as a perfect candidate for more extended research efforts. However, current architectures of programmable controllers' software and hardware do not fit to the idea of multiagent control. The next step towards practical application of multiagent approaches needs to address this issue by proposing and testing the corresponding low-level architectures for automation systems. This paper presents such an attempt.

This paper is structured as follows. Section II identifies main problems which need to be solved in order to address the challenges of BHSs automation by applying multiagent holonic control. Section III reviews relevant developments in the area of intelligent automation, such as: the IEC 61499 architecture, object-oriented engineering and applications of multiagent and holonic systems. Section IV presents the developed framework for intelligent BHS automation, which is based on the IEC 61499 standard. Section V focuses on the intelligent controller functionality. Section VI further describes the distributed baggage path planning implementation. Section VII presents trial implementations and case studies. This paper is concluded with the summary of results and an overview of future developments in Section VIII.

II. Problem Statement and General Approach to Solution

Multiagent approach to automation of BHSs seems to be a promising solution for improving their flexibility of operation and efficiency of their design. This hypothesis, however, needs to be confirmed in case studies of realistic complexity. New hardware and software architectures for the low level of automation systems are required to use holonic control systematically in industry.

This paper proposes new architecture of an embedded intelligent control implementation with IEC 61499. It aims to show that many of the requirements for building holonic agents are inherent in the IEC 61499 specification. The proposed architecture aims at implementation of holonic control directly on embedded devices. This represents a step toward industrial application of intelligent automation principles.

The central part of the proposed architecture is a reusable intelligent software component for baggage handling encapsulated in an IEC 61499 function block. This enables easy deployment of the developed application on arbitrary topologies of networking controllers.

The new degree of BHS flexibility is achieved on account of collaborative behavior of intelligent controllers. The intelligence is achieved by applying distributed baggage routing algorithms, combining simulation, real-time control, and predictive control. It is demonstrated that the proposed architecture can support efficient reconfiguration of the BHS, in terms of changing its physical layout, or by changing the number and interconnections of embedded controllers. Flexible visualizer is created for viewing state of the BHS models in simulation or in real-time operation.

III. Related Works

A. Distributed, Multiagent, and Holonic Approaches to BHS Automation

The modular nature of material handling systems has inspired some researchers to try their distributed automation, such as work [9], where each conveyor is controlled by an embedded device with wireless communication capabilities.

Application of a multiagent approach to baggage handling was presented in [8], where a Java application was implementing JADE-based agents communicating via FIPA-ACL agent communication language [10]. The authors describe successful agent-based implementation of a variety of baggage handling control actions under simulation. They also describe that the limiting factor for the performance of the system was the messaging overhead of the agents' communications.

Many current research projects into multiagent control systems e.g., [8], [11], and [12], start with implementation of a general purpose agent that is capable of executing arbitrary behaviors. Most of these applications, however, aim at offline application (simulation), or require a multilayered hardware architecture, where the low-level control tasks are still implemented in PLCs, while the agent behavior is running on a separate powerful computer. This, naturally, restricts wide application of multiagent control in the automation practice.

B. IEC 61499 Architecture as the Next Generation of PLC Technology

Addressing the limitations of the legacy PLC programming languages and looking toward the realities of implementing real-time multiagent systems, the International Electrotechnical Commission (IEC) initiated a project to encourage the development of new software architecture, extending the IEC 61131-3 Function Blocks by adding event driven execution. In 2005, this project culminated with the approval of the IEC 61499 standard [3] that defines the new function blocks architecture. Unlike previous standardization efforts, this is not a retrospective recognition of practices, but an attempt to guide future developments toward an open standard that allows genuine vendor interoperability.

At one level, function blocks provide a direct advance from, and viable replacement for, established automation programming languages such as ladder logic, structured text, or their proprietary variants. However, their application extends past simple replacement of legacy systems because of the inherent support for distributed applications and the ability to provide a platform for modeling and simulation with well-defined interfaces.

There is a small but growing toolset for function block design. The Function Block Development Kit (FBDK) [13] remains the most widely used, because it is the oldest and is free for educational use. Commercial tool support is also beginning to emerge. The new version of the ISaGRAF industrial control design software with support for IEC 61499 Function Blocks is introduced in [5].

In order for function blocks to become executable on a variety of hardware, hardware vendors must provide support for the standard. The options remain limited, but are on the increase.

There are currently several options for executing function blocks. First, any platform that can execute standard Java byte code can run the FBRT [13]. This includes desktop computers running any major operating system. Embedded execution option includes the Elsist Netmaster II, which runs a cut-down version of Java Standard Edition (J2SE). Tait Control Systems MO'Intelligence units run Java Micro Edition (J2ME) and are supplied with a port of the function block runtime and vendor supplied Service Interface Function Blocks for hardware access. These units are available in several formats with support for DeviceNet and an integrated motor drive option.

C. Efforts on Improving Engineering Efficiency of Automation Systems

There exist a multitude of attempts aiming at improved efficiency of the engineering and reengineering process of automation systems. These are generally categorized along the continuum of abstraction versus implementation. That is, the more abstract methods, such as Unified Modeling Language (UML), are usually more able to describe a broader range of systems, while the implementation focused methods may be directly executable, but are too specific to be of general use.

The IEC 61499 architecture seems to offer quite optimal abstraction/implementation ratios. Function blocks are one framework that promises the ability to break out of the purely implementation phase, allowing a designer to build applications whose structure mirrors that of the physical systems with which it interacts, while still being directly executable. A number of research projects both in academia and in industry, e.g., [14]–[16], describe architectures that aim to solve particular challenges in the design and deployment of distributed control and each specify function blocks to a greater or lesser extent. As it is pointed out in the survey [17], the common factor across most design methods is the attempt to use object oriented (OO) techniques applied to function blocks.

One example of these is the concept of an "automation object" (AO), explored in [18], [19] and, in particular, in [20], which defines it as "*a collection of data and knowledge elements belonging/relevant/describing physical building blocks of automated manufacturing system*." The AO concept extends the modularity of software or hardware to the modularity of the whole entity, which combines mechanical, electrical and software components into intelligent mechatronic devices.

In [19], it was concluded, that IEC 61499 is an appropriate architecture to organize the *information technology* (IT) side of AOs. One such feature of IEC 61499 is the definition of interfaces via adapter interface function blocks, which makes it possible to design function blocks that can be readily substituted for one another, as demonstrated e.g., in [21]. This contributes to the rapid reconfiguration of applications which is increasingly a requirement for automation technologies and IEC 61499. The architecture presented in this paper makes extensive use of adapters to minimize the number of connections required, and to allow reconfiguration at design time.

Currently, there are several groups working on creating AO architectures incorporating function blocks as a major part, for example, [22]. In [23], the idea of intelligent machines is extended using the example of conveyor systems and provides additional reasons for the use of the function block architecture.

The general approaches toward combining mechanical systems with electronics and software to create complete reusable mechatronic components, differ in their focus and scope. On one point, all discussions appear to agree: that a key aspect of achieving multidomain modularity is to allow the logic of the control application to be organized in the same way as the physical system being controlled. This seems sensible—the encapsulation should be consistent across the mechanical, functional, and logical domains allowing true modularity through the complete model.

D. Systems Modeling and Simulation

Another benefit of using IEC 61499 as a modeling language is that it is directly executable, so it can readily be used for simulation. This allows a modeled system and accompanying control system to be tested before deployment. This would constitute a serious advancement compared to the state-of-the-art, where simulation is used only for general system prototyping at early stages of development. In most cases, the behavior tested via simulation then needs to be implemented in the controller of the BHS, and this is very resource-consuming and error prone process. Once the controller is developed, its verification by simulation would also require extra development effort.

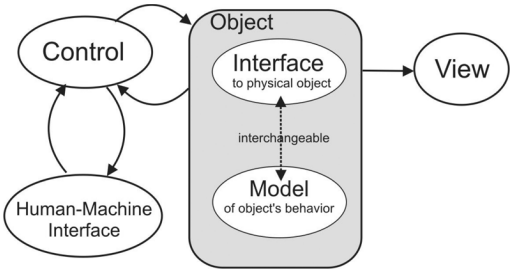


Fig. 1. Model-view-control architecture in automation.

In [24], Christensen describes the application of the model-view-control (MVC) design pattern to function block system design, providing the foundation for the internal structure of AOs capable of immediate simulation. The diagram in Fig. 1 illustrates the pattern, which is based on the observation of similarity of the interfaces of the real physical object (say, conveyor) and of its simulation model. Thus, these components (model or interface) can be used interchangeably, being represented to the outer world by the object interface. The view component is "fed" by the parameters generated by the object and renders its current state. Finally, the controller is connected in closed-loop with the object, receiving from it readings of sensors (either real or simulated) and sending it control signals. The human-machine interface component supports manual control of the controller and rendering of its status. The corresponding design methodology, exemplified in [20], suggests to start controller development and testing by connecting it in closed-loop with the model and ensure its validity by simulation. Then, the model component is to be seamlessly substituted by the interface to real sensors and actuators. Various examples of the system design combining MVC and IEC 61499 are accompanying the FBDK. Rockwell's MAST simulation platform [11], suggests the use of function blocks for the low-level control, working under the direction of a supervisory software agent that manages connections with other agents. However, it seems that no practical experience towards this end was gained in that work.

Furthermore, if simulation is performed in function blocks, these same blocks may be deployed into the final system where predictive control behaviors are required. Hirsch *et al.* in [25] describe the use of physical modeling and simulation to assist in designing a control system. It also suggests that the scheme could be extended to include simulation in the control system itself to provide simulated prediction of the system. This technique is a variant of "model predictive control," a well-established principle in control systems, where a mathematical model of the plant is used to predict its behavior into the near future.

E. IEC 61499 for Holonic and Multiagent Systems

Application of function blocks for building holonic systems has long been envisaged. Thus, the Holonic Manufacturing Systems project [4] suggested the use of function blocks from early stage [26], the ideas were further specified, for example, in [27] and [28].

Numerous design methods, architectures, computing platforms, networking technologies, and programming languages have already been proposed to help improve automation systems using a multiagent approach, some incorporating the use of IEC 61499 Function Blocks [28], [29]. In [21], the AO concept was used to create intelligent mechatronic devices using IEC 61499 features coupled with agent-based control.

These works form the necessary critical mass for proposing a solution combining holonics and IEC 61499 in BHS.

IV. IEC 61499—Based Architecture for Intelligent BHS

The approach taken in this work is object-oriented in the sense that the structure of software mimics the structure of the physical BHS and is centered around the conveyor mechatronic component. A reusable software component (function block) represents a single conveyor in the BHS control system, which is composed of as many such function blocks as conveyors in the physical system. This approach stems from [21], where a bottom-up approach is taken to the challenges of mechatronic modeling. It is applied in BHS and extended by autonomous, agent-based behavior following [30].

A. The Conveyor Model

The approach taken in this model is that the primary software component will represent one section of conveyor including its various sensors, actuators, computing platform, and control software. This is a reasonable tradeoff between flexibility and maintaining simplicity in the design where these blocks are to be deployed. The approach taken is a little different from [11], where conveyors are modeled as assemblies of services such as belts, diverters, and scanners at the same level.

We begin with the development of a generic conveyor model by identifying typical functions and interactions of a single conveyor. Conveyor-based BHS are constructed of a set of "conveyor sections" connected end-to-end, or in merge or divert configurations. If we imagine a fully featured conveyor component, able to perform any of these actions, it would look like the general purpose conveyor section shown in Fig. 2. This contains the mechanical conveyor components required for merge and divert, the sensors for detecting bags and measuring belt speed, and a motor with drive to make the belt move. It also includes an embedded controller that makes control decisions based on sensor data and from information exchanged with other conveyor controllers connected by network.

The generalized conveyor section of Fig. 2 is the initial model for a reusable component that could describe a section of conveyor at several levels from its logical connection to other sections, to its dimensions and other physical parameters. It was desired that a network of conveyors could then be modeled by simply making connections between appropriately parameterized conveyor function blocks.

The design follows the extended MVC design pattern. In this work an extension of this pattern, called Predictive Object-View-Controller (POVC) has been developed and tested. Instead of switching between model and real object during design time, they both are combined in one component. Fig. 3 shows internal architecture of the conveyor software component built according to the POVC pattern, and exemplifies interactions between components representing two conveyors. The Object composite component includes both the model of dynamics (including simulated sensors), and the interface to

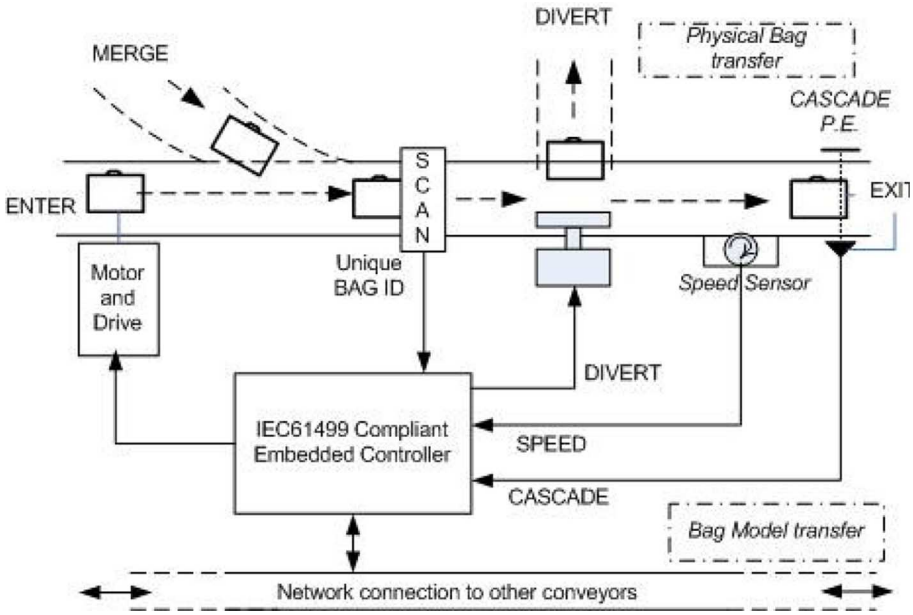


Fig. 2. A fully featured conveyor module.

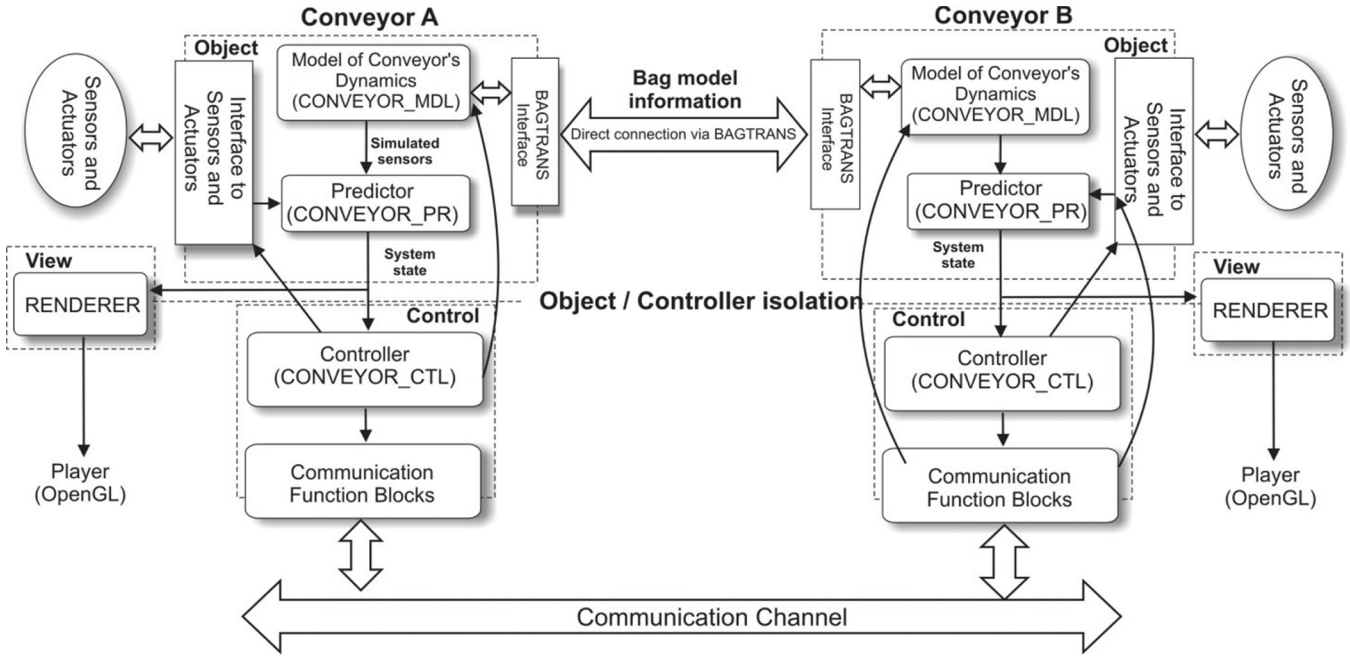


Fig. 3. Internal architecture of the conveyor component following extended MVC with illustrated interactions between models of two conveyors (simulation configuration).

real sensors and actuators. Depending on the mode of operation (online control or offline simulation) the Predictor module delivers to the Controller values of actual or simulated sensors. The simulation keeps running even in the on-line operation mode, in this case if readings of real sensors are temporarily not available, e.g., due to a malfunction, the Predictor will use the simulated ones.

Models of adjacent conveyors exchange the bag model information via BAGTRANS interface which will be discussed further in Section IV-C. In this way a model of the complete BHS can be created as interconnection of the models of constituent conveyors, synchronized and communicating via the BAGTRANS interface.

The View component sends the current state information to the standalone visualization application, which renders the current state of the whole conveyor system as discussed in Section IV-F.

The control part, along with the Controller, includes communication function blocks enabling intercontroller communication, required for implementation of such distributed intelligence features as dynamic baggage routing. Details of the intelligent control implementation are presented in Sections V and VI.

B. Complete Conveyor Component

The corresponding function block implementation has a multilevel structure where the low-level operations are wrapped

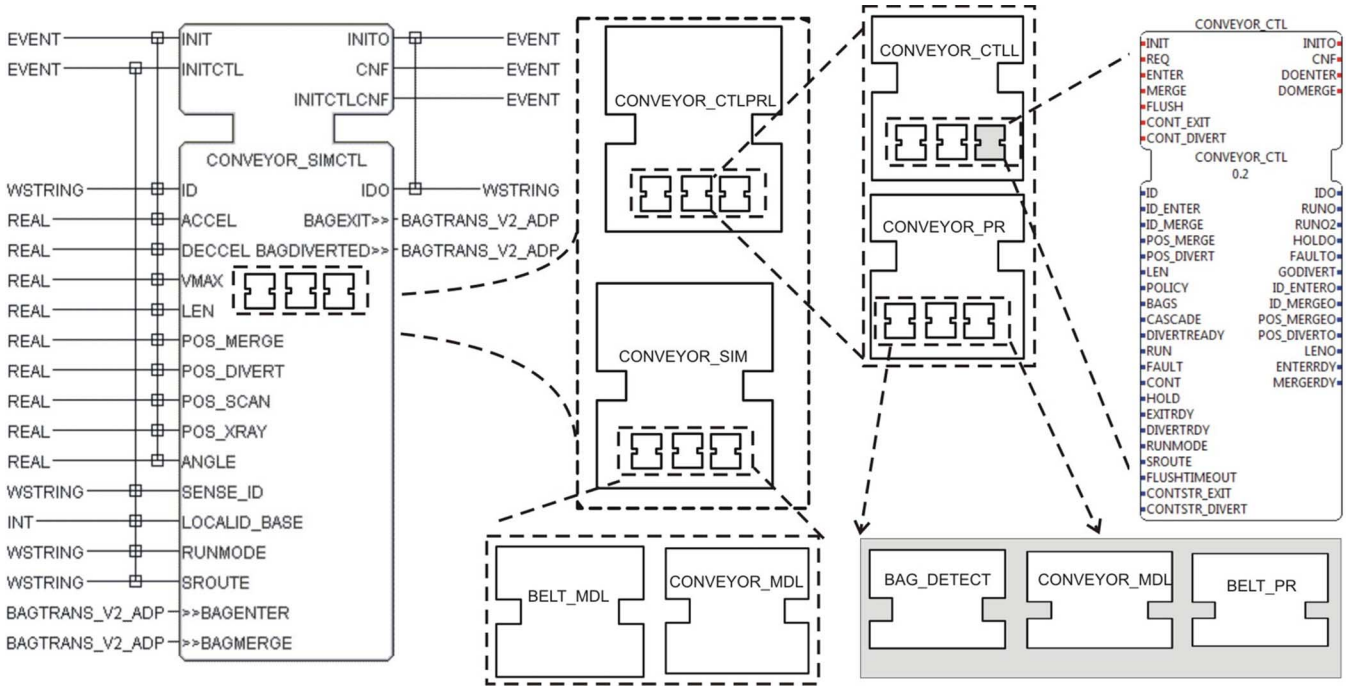


Fig. 4. CONVEYOR_SIMCTL interface and internal structure.

in composite blocks that hide details, provide connectivity and present a clean interface for the designer to create system models without detailed knowledge of low-level functionality.

The top-level function block, CONVEYOR_SIMCTL, shown in Fig. 4 has all the functionality required to define conveyor connections and layout, simulate the network and demonstrate distributed control of the simulated network.

The following is a summary of the actions that can be performed by a conveyor, although not all of these can be executed by the same section. For instance, the model will not allow a section to implement both merge and divert due to possible complications with the path planning system.

- **Transport**: The ability to move objects from one end to the other.
- **Detect**: Each conveyor is equipped with a cascade photo eye (PE) at its end and a motor drive. Conveyors that implement divert have an additional PE at the divert point.
- **Merge**: Allow bags to merge into this conveyor from others. The merge point can be positioned anywhere within the length of the section.
- **Divert**: The ability to eject bags from the stream of bags into another conveyor. Like merge can be positioned anywhere in the conveyor.
- **Scan**: The scanner can read the details of a bag including its globally unique ID and its required destination in the BHS.
- **X-ray**: The X-ray is responsible for determining the security status of the bag, which may determine whether the bag is eligible for delivery.

The CONVEYOR_SIMCTL function block packages a significant amount of functionality. In order to give a general sense for the hierarchy involved, Fig. 4, shows a simplified depiction of the important function blocks required to perform simulation and control. Functionality of some basic blocks is as follows.

CONVEYOR_MDL is the primary engine for simulation of bag behavior, implementing the functionality for inserting and removing bags from a conveyor section model and predicting bag positions through time.

BELT_MDL abstracts the lowest level physical behavior of the conveyor by simulating the dynamics of a conveyor belt with inertia. It also simulates the behavior of a rotary encoder, producing an event each time the belt moves by a preset distance. This encoder output is the signal seen by the control system.

CONVEYOR_PR combines the simulation component CONVEYOR_MDL with bag tracking logic to provide a real time estimate of the location of bags within the conveyor section.

The BAG_DETECT block encapsulates the task of detecting and measuring bags as they pass the PE on the conveyor.

C. Modeling a Bag

The conveyors make up the fixed part of the system, while the bags are dynamic, being passed between conveyors. Together the conveyor components plus the bags represent almost the complete BHS system.

Being a distributed control system, the data representation of a bag takes on rather great importance. The physical (or simulated) bag must be accompanied in the control system by a data representation containing all information necessary for its correct processing. The bag-related data were encapsulated in an IEC 61499 custom data type. This is a simple data record containing an arbitrary number of IEC 61499 primitive types, accessible by name. When first detected the bag has only an auto-generated local ID. The remaining information must be obtained as the baggage handling process executes by the control system making use of scan and X-ray facilities.

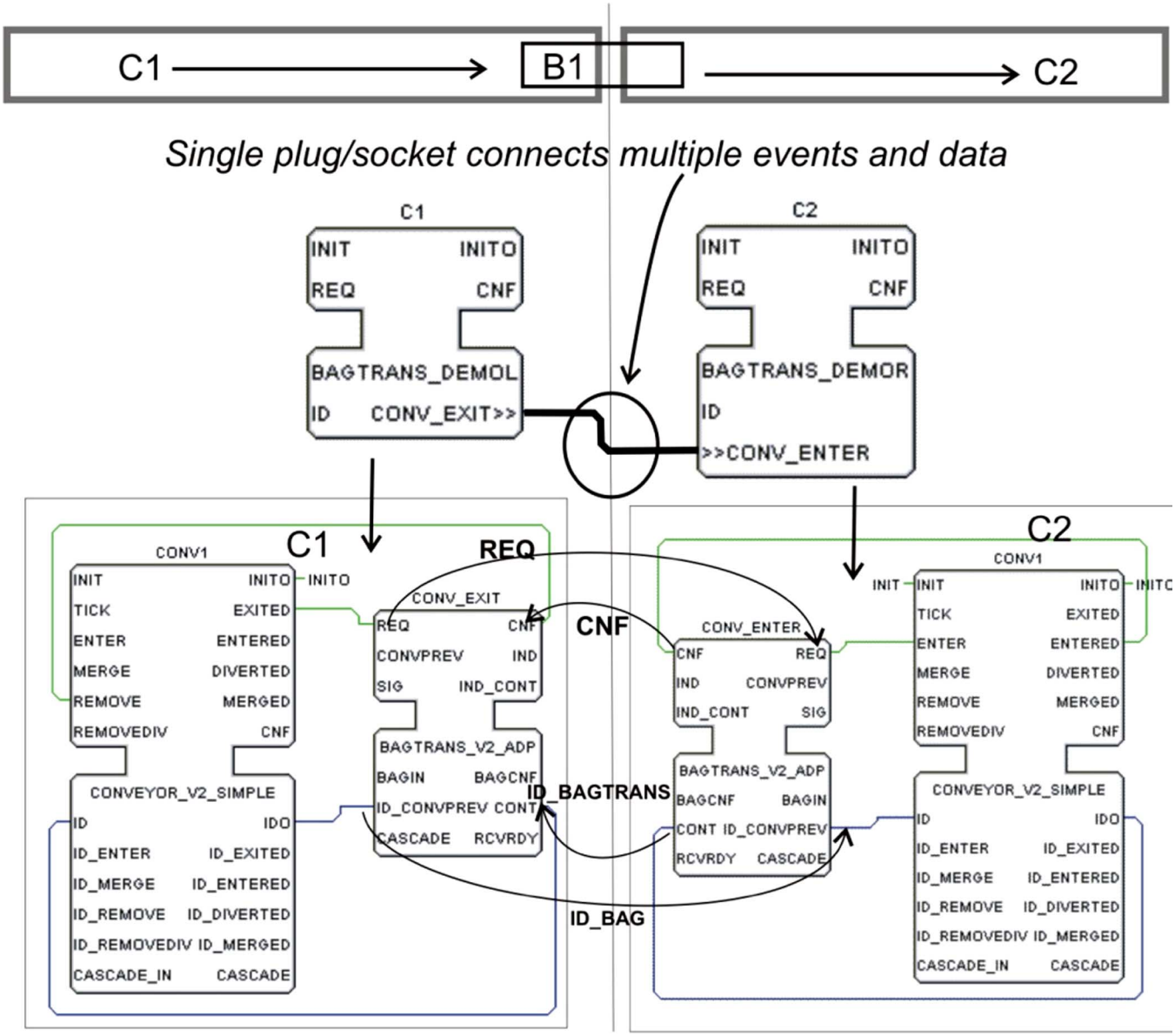


Fig. 5. Adapter-based connection one line between models of two conveyors and the logic of bag transfer.

D. Connecting Conveyor Sections

The CONVEYOR_SIMCTL function block introduced above represents a section of conveyor and its functionality. For truly distributed execution, each conveyor must track any bags within its length and make control decisions about their management and delivery.

Whenever a bag moves from one conveyor to another, there is a need to also communicate the data representing that bag to the downstream conveyor, and to remove the record of that bag from the upstream. This is simplified by encapsulation of the bag data into a single custom data type, as shown above, reducing the number of data signals required. However, each conveyor must have the ability to indicate readiness to receive so that upstream bags will only be supplied when appropriate, while a further event and data pair, IND_PATH and PATH, responsible for building possible delivery paths must also be connected. As a result, there are a number of events and signals to be connected to allow bag transactions to occur. The need for manual creation of many connections is time consuming, error-prone and clearly at odds with the stated goal of achieving an easy process of modeling within the FBDK environment.

To simplify the connection of conveyor sections, CONVEYOR_SIMCTL, makes use of IEC 61499 adapters.

Although adapters appear much like other function blocks, they are simply an interface definition containing no functionality. Adapters can assist in allowing reconfigurable applications, in the engineering view by minimizing the visible connections and also at runtime by reducing the number of management commands required to add, remove or modify a relationship between blocks. More details on the benefits of adapters in function block designs can be found in [20, Ch. 16].

Fig. 5 demonstrates the transaction of a bag B1, exiting from Conveyor C1 on the left to Conveyor C2 on the right. C1 and C2 are very simplified conveyor function blocks showing only the required bag transfer logic. When a bag reaches the end of

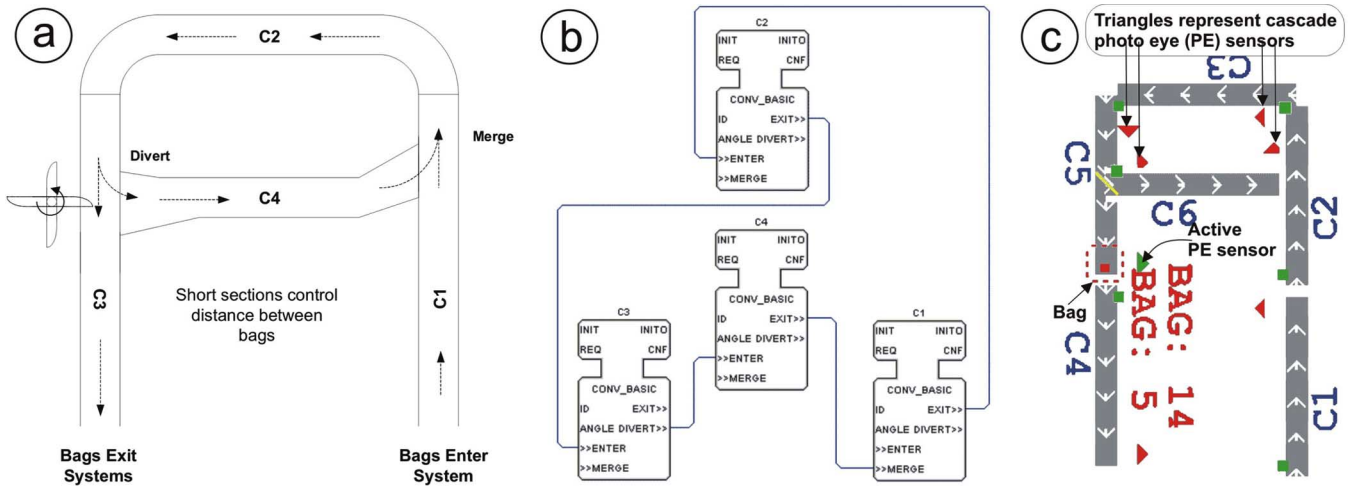


Fig. 6. (a) Example BHS fragment for modeling. (b) Function block model of the conveyor chain. (c) Sample visualization of the conveyor system.

C1, the EXITED event is triggered and the BAG_ID output is set appropriately. This event is directed through the BAGTRANS adapter to the ENTER event input of C2. If C2 is able to accept a bag then it will respond by triggering the ENTERED event which connects through the adapter back to the REMOVE input of Conveyor 1 indicating that the bag is to be removed from C1. Note that this shows only the transfer of a bag model, not the control sequence by which it is negotiated.

E. Modeling a Baggage Handling System (BHS)

Having introduced the conveyor software component and explained the mechanism for interconnection, we can now look at how these components can be assembled to create a model of a conveyor BHS.

Let us take the simple BHS fragment shown in Fig. 6(a) and examine how it can easily be modeled using a derivative of the CONVEYOR_SIMCTL function block with each block representing a single conveyor section. This example contains four conveyor sections labeled C1 to C4. There is a divert path that leads from C3 to C4, and C4 merges into C1.

To simplify the example, the CONVEYOR_SIMCTL blocks are encapsulated into CONV_BASIC blocks that hide much of the detail of the CONVEYOR_SIMCTL block by assigning default values to most parameters. The structure of the network is defined simply by the connection of the ENTER, EXIT, MERGE, and DIVERT ports. Fig. 6(b) shows a part of application that models the chain from Fig. (a), laid out similarly to the original example and with the same conveyor labels. Note that the model as shown is simplified and offers no way for bags to enter or exit the system.

By creating the function block model of Fig. 6(b), the designer achieves the following.

- Specified the physical dimensions and layout of the conveyor sections (it is assumed that the CONV_BASIC block predefines conveyor length).
- Created the basis for simulation of the system.
- Created a distributed control system for the system.

F. Visualization of Conveyor Models

To provide intuitive information about a conveyor system as represented by a function block network, it is very useful to have some method of displaying the network in a way that resembles the real conveyor system. The FBDK contains building blocks for simple visualization of common industrial system components such as solenoids, linear actuators and drills. The representation of the conveyor network was seen as limited by the FBDK's visualization components because while the conveyors are generally static, the bags they carry are dynamic and will vary in number per conveyor. To avoid these limitations a simple OpenGL-based visualizer was created that runs as a standalone application outside the FBDK.

The visualizer directly parses the XML system configuration, which allows rendering of the static layout of conveyor sections without execution of the function block system. Once activated, simple network datagram packets are generated by the CONVEYOR_MDL blocks representing the state of the conveyor section and the details of any bags present. These packets are used to display the conveyor sections and bags in real-time, including the state of sensors and actuators.

Fig. 6(c) shows a sample visualization of the conveyor network modeled by the system configuration of Fig. 6(a). In this example, a bag is shown traversing Conveyor C1. The triangles represent the Cascade PE sensors, the end of each section, and the white arrows show that a given conveyor is running and the direction of travel.

V. Intelligent Collaborative Baggage Handling Controller

A. Structure of the Controller

In the proposed architecture control of the BHS is achieved via collaborative effort of the controllers of single conveyor sections without any central supervisor. As a result, the control is adaptive to the layout and status of the BHS. For example, it can dynamically change the routes of baggage delivery in case if some conveyor section is out of operation.

There are two main parts to the control system. First is simple reactive control, where a set of rules are applied according to the position of bags, the state of sensors and the state of flags from downstream conveyors. The second, directing these simple reactive control actions is a path planning controller that is respon-

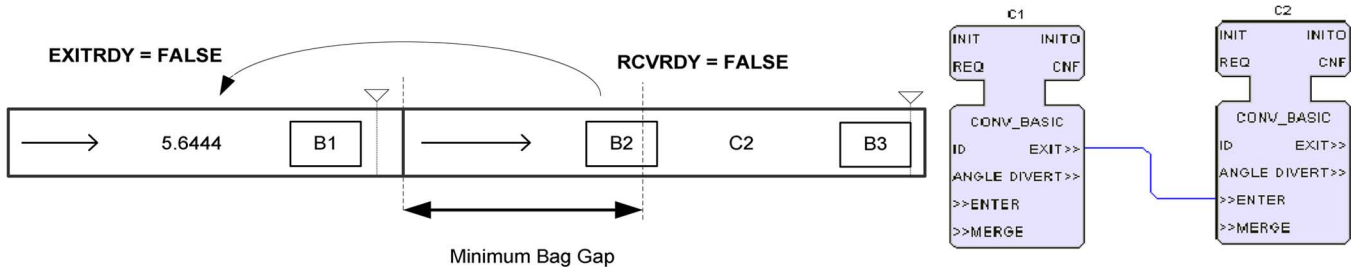


Fig. 7. Bag spacing control.

sible for actually guiding bags to their destination. The latter will be discussed in Section VI.

The collaborative "agent" behavior of the conveyor is performed by the controller function block CONVEYOR_CTL, which executes the actual decision making part of the BHS. Its interface is presented in Fig. 4. The controller is implemented as a basic function block which combines the reactive behavior programmed in ladder logic with higher level functions programmed in Java.

B. Reactive Behavior and Interfaces

As with all interactions between conveyors, signalling between controllers occurs through the BAGTRANS interface. Each conveyor produces ENTERRDY and MERGERDY signals, which are received by upstream conveyors into their EXITRDY or DIVERTRDY inputs depending on conveyor layout.

Using these signals the controller is able to implement a set of control actions for managing bag traffic. The major limitation is that the controller only has direct information about its own conveyor section and limited data from adjoining sections, whereas a centralized controller may freely use information from throughout the network. This did not prove to be a problem for the control actions implemented so far.

The simplest required behavior is the ability to observe and obey EXITRDY and DIVERTRDY signals as requested to prevent exiting bags from colliding with others or getting so close as to cause problems with tracking. This behavior requires only the flags from downstream conveyors and the value of the Cascade or Divert PE to operate. To make this behavior useful, the ENTERRDY output signal (which supplies the EXITRDY or DIVERTRDY inputs) must be generated by the downstream conveyor. This is done simply by searching the BAGS array input from the predictor to determine is a sufficient gap exists for a new bag to be received. The MERGERDY signal is generated in a similar way, but the gap search is performed at the merge positions indicated instead at the conveyor entrance.

C. Cascade Stop and Bag Spacing

To prevent bags getting confused or misdirected, it is important that a reasonable gap is maintained between each. This can be achieved by an upstream conveyor stopping to put extra gap between bags. This behavior is achieved by simple Boolean flag interactions between conveyor sections as specified by the BAGTRANS interface. Each conveyor determines whether it can receive additional bags by searching its bag list for any bags near the conveyor start. Fig. 7 shows C2 is not ready to receive because bag B2 is still in the "minimum gap" region.

C1 can only run until B1 triggers the cascade PE and then must wait. Although the EXITRDY input is generated remotely and may be received via a network channel, this behavior can be expressed, for example, in a simple ladder logic statement, which is a supported IEC 61499 algorithm language. As long as the cascadeStop flag is true, then the conveyor will not run. Other control behaviors are implemented in a similar fashion. This also demonstrates that legacy languages, well suited to the expression of simple automation concepts, are easily applied within a Function Block design without the disadvantages being committed to these languages for the whole design.

D. Merge Control

When two streams of traffic merge, it is necessary for one stream to wait for a gap in the other before delivering each bag. This is one of the more important aspects of bag management in a BHS as incorrect merging can easily result in collisions or misordering of bags resulting in lost bags. In the function block controller, it is implemented in a simple manner using the MERGERDY signal to indicate readiness for merge traffic. This is a very simple approach to the problem and may need additional treatment in future enhancements to the controller.

VI. Dynamic Path Planning

In order for a BHS to be useful, each bag must be delivered to the appropriate destination. The function vividly demonstrating the distributed intelligence potential of the proposed BHS architecture is decentralized planning of the path for each piece of baggage. The path planning results in the decision of bags' divert so that they take the correct path.

In conventional conveyor control systems this is done by a central routing controller that has a complete model of the network layout and can perform a tree search of possible paths between points using established path finding algorithms. The method described here demonstrates that same or even better global behavior can be achieved in the distributed architecture with fully decentralized control logic.

A. Dynamic Path Building

A series of connected conveyors can be modeled formally as a weighted directed graph. That is a graph of nodes connected by vertices with associated "costs." The problem of finding an optimal path is that of finding the sequence of graph edges that incurs the least total cost. When applying a centralized control system, algorithms for determining the shortest path, such as

Dijkstra's [31], can readily be applied to the model to give a result.

In the case of distributed control, no individual part of the system has a complete view so as to allow a global analysis to be made. As a result, possible paths must be determined in a cooperative manner by propagation of messages along the same routes that are available to the modeled bags. The algorithm used to determine least cost path from a conveyor to the baggage destination is similar to the distributed Bellman–Ford algorithm [32]. Unlike the original algorithm, which calculates shortest paths to all graph nodes by maintaining and updating vector of distances, our algorithm generates path strings to all possible destinations. This helps to add specific processing features to the bag's itinerary and avoid undesired loops.

The technique described uses the interconnections between conveyor sections to construct valid paths from each conveyor to each reachable exit using back propagation. When requested, destination or "exit" points in the network emit PATH signals to their predecessors. These append the path signal to their own identity and the branching details (merge, divert, enter, exit) and send them on in turn to the upstream nodes. This builds path strings for each (reachable) network node that define all the possible future choice combinations from that node. Working back from the network exit points allows each node to only store the minimum path information to reach each available exit. When this propagation process reaches source or entry points in the network then complete paths are defined. These path strings can be regularly regenerated to recognize changes in network topology, such as component failure or even reconnection of conveyor sections.

To slightly complicate matters, simply finding a path through the conveyors to a location is not sufficient. In most cases, the bags will be initially unidentified and have no security clearance. The path planner must route bags through the BHS so that they are identified by a scanner and X-rayed for security before finally determining a path for their destination. The action of integrating additional processing actions into the path planning system suggests the idea that the scheme could be used for a far more general class of materials handling and processing sequences.

The internal representation of these paths currently uses strings to allow easy communication between conveyors. A simple format is used to represent the possible actions of each conveyor. The form of these strings is a concatenation of conveyor IDs tagged with actions that have been propagated back from a destination. These actions may be any of those supported by the conveyor component. A separate entry is required to enter a conveyor, perform any internal actions and to exit the conveyor. To keep the length of path strings manageable, actions are abbreviated, as shown in Table I.

By joining a sequence of these actions a complete sequence can be described including both physical path and processing actions.

TABLE I
PATH STRING ABBREVIATIONS

Action	Path String Abbreviation	Example
Exit	x	x(A) exits conveyor A
Merge	m	m(A) merge to conveyor A
Divert	d	d(A) divert from conveyor A
Enter	e	e(A) enter conveyor A
Scan	i	i(A) scan (identify) at conveyor A
X-ray	s	s(A) perform X-ray security test at conveyor A

B. Loop Detection

Most airport BHS contain loops. These are useful for reprocessing mishandled bags, and may also be used to change the order of bags in the system, for instance to implement a rush bag feature. These operations are currently not supported by the path planning system, but it is still necessary to be able to detect the presence of loops in a system under analysis.

The existence of loops presents a problem for the path builder which would never terminate as it continues to propagate around the loops. To prevent this, the path builder must incorporate a loop detector. This applies the rule 'if taking option A led to a repeat in the formula, do not reattempt option A'. The requirement for the loop detector does add considerably to the processing load required for continuously updated path analysis. However, it is not so burdensome if applied only at initialization or if explicitly requested for network reconfiguration or for component failure.

A special case is that what appears to be a loop may not, in fact, be so if it contains a processing action such as a scanner or X-ray. This is because the first trip through a loop containing a scanner leaves bags modified (i.e., identified) after the completion of the loop meaning that bag's "position + status" in the overall system is not the same as before entering the loop. The example below demonstrates this point.

C. Example of Path Building

Fig. 8 shows a simple path building example including a loop. X is the entry point and Y is the destination, A and B are simple straight conveyors, while C is a loop section and includes a scanning station. The numbers indicate the sequence in which messages back-propagate from destination to source and the process is summarized including the step by step path string construction.

Note that at step 5 there appears to be a loop at B, however, because a scan has been encountered the loop detect is not triggered. The loop is finally observed when the path building repropagates to C a second time at step 8, completing the analysis.

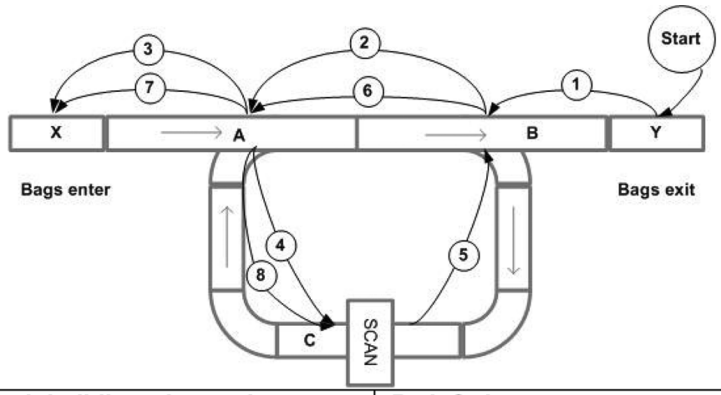
Because the system contains a loop, the source receives two possible paths. The first is direct through A and B described by

$$\mathrm{e(A)} \times \mathrm{(A)e(B)} \times \mathrm{(B)} -> \mathrm{Y}.$$

The other looping through C which includes the scan is

$$\mathrm{e(A)} \times \mathrm{(A)e(B)d(B)i(C)} \times \mathrm{(C)m(A)} \times \mathrm{(A)e(B)} \times \mathrm{(B)} -> \mathrm{i(Y)}.$$

Note that the destination i(Y) indicates that this path will reach the destination Y having scanned the bag.



Path building triggered	Path String
1. B notified can deliver bags to Y	x(B) -> Y
2. A notified can deliver bags to Y via B	x(A)e(B)x(B) -> Y
3. Start point X has valid path to destination Y	X.e(A)x(A)e(B)x(B) -> Y
4. Alternate path: C can scan and deliver bags to Y via A	i(C)x(C)m(A)x(A)e(B)x(B) -> iY
5. B can scan and deliver bags to Y by diverting to C	d(B)i(C)x(C)m(A)x(A)e(B)x(B) -> iY
6. A can scan and deliver bags to Y via B	d(B)i(C)x(C)m(A)x(A)e(B)x(B) -> iY
7. Start point X now has new path to Y with scan	X.e(A)x(A)e(B)d(B)i(C)x(C)m(A)x(A)e(B)x(B) -> iY
8. re-Propagation to C – Loop detected path building stops	+ X.e(A)x(A)e(B)x(B) -> Y

Fig. 8. Example of path building.

D. Applying Path Strings and Directing Unidentified Bags

The result of this analysis from Fig. 8 is that the path planner knows that bags entering at X have one way of getting to Y and one way of getting to i(Y). Once the controller has received valid path strings from reachable destinations, the task of routing become simply that of identifying the immediate downstream conveyor in a valid path that terminates with the required destination.

When bags enter a BHS they are unidentified and their destination is unknown. There is also the possibility that bags become misplaced during processing and separated from their model in the tracking system. As a result, the controller has a default behavior that any unidentified bags will be routed to the nearest scanning station to allow them to be properly directed.

VII. Trial Implementations

The trial implementation of the proposed function—block architecture aimed at several goals, some of which are as follows.

— Check feasibility and correctness of the proposed decentralized algorithm for dynamic path planning.
— Test the performance of the underlying Java-based implementation of function blocks for running offline simulation scenario.
— Test the performance of the FBRT running on embedded control device and feasibility of distributed control of BHS w.r.t. performance and reliability;

A. Tests on a Simulated BHS Layout

These tests aim to demonstrate some of the more complex features of the BHS controller that could not be performed on the real system at this stage. A basic set of realistic tasks is executed to determine if the controller is able to process bags correctly. To improve the realism of the tests, some trials are run where the simulation is modified to introduce random perturbations in the system behavior including bag slippage and sensor and actuator failure.

The BHS to be simulated for these tests is shown in Fig. 9(left). Its complexity corresponds to a typical small airport. The process represents a simple but typical check-in process for departing bags. The process begins at the check-in counter where bags are weighed, assigned a temporary unique ID number and tagged as belonging to the particular passenger.

The bags then progress along a conveyor to the induction point, where the baggage handling control system becomes aware of the physical bag and begins to track its progress. Shortly after induction is a scanner. This reads the tag affixed at check-in allowing the control system to link the tracked bag with a unique bag identity. After scanning, bags move to the X-ray stations. Following the X-ray stations, a bag should have one of three states:

- X-rayed and cleared: the bag can progress toward the aircraft;
- X-rayed and flagged for further manual inspection;
- For some reason the bag did not pass through the X-ray, or its state was not recorded.

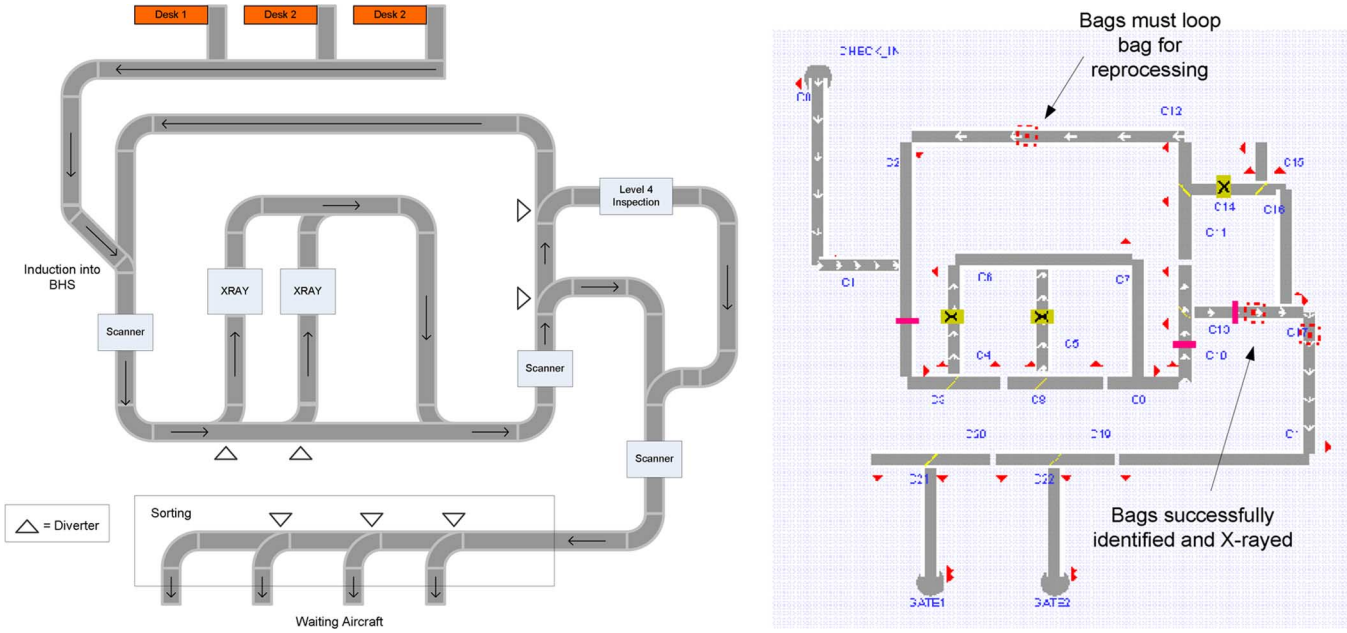


Fig. 9. Layout of a small airport BHS and test "two closely packed bags" visualized on the rendering application.

In the second case, bags are flagged as requiring further inspection (called Level 4 Inspection) in which case they are diverted to the L4 station. The third situation where a bag has not received a status from the X-ray station is handled by simply sending the bag around the main loop for another attempt. Finally, cleared bags enter a sorting system that checks the bag identity once more, and delivers them to the correct aircraft based on their tagged destination.

To produce a repeatable test for the simulated control system, a method was required to generate a predetermined input test sequence. This consists of a series of bags being fed into the check-in system with particular spacing and attributes so as to test the various aspects of the system, along with an accompanying trigger of specific failure modes within the simulation. A composite block was created to trigger the parameterized bag production and produce the failure trigger events. The exit nodes at the destination gates record which bags are received, allowing a review of delivery success.

At this stage, results from tests are difficult to quantify because they cannot yet be compared to a real BHS, however, the ability to conduct reproducible tests provides a starting point for validating general functionality of the simulator and control system. Furthermore, these tests can form a basis for evaluation of any subsequent improvements to the design.

The automated test function block was used to supply preset sequences of bags to the system to measure its response. The following tests were conducted: (i) alternating destinations; (ii) recognition of closely packed bags; (iii) conveyor failure; and (iv) scanner failure.

For example, in the test (ii) "Closely packed bags," three bags are supplied that are too closely spaced for proper tracking. The system must separate them and deliver correctly.

Fig. 9(right) presents a visualization snapshot of the BHS during the test. All bags are eventually delivered correctly. This is partly due to the fact that the simulation only diverts one bag per divert trigger, making it fairly inevitable that the bags become separated. Once apart, the tracking system can readily detect and identify the individual bags. The first two bags are processed normally, but the third fails to divert for X-ray and is recycled around C12 to rescan and perform X-ray before sorting.

B. *Tests on a Laboratory Testbed*

The current test setup is based on the Festo MPS500 conveyor belt loop consisting of four straight sections, designed to transport work pieces or pallets. The loop has six "stations" at which point there are sensors to detect the presence of work pieces. Each of the four sections is driven by a geared three-phase motor.

The originally PLC-based control hardware of the MPS-500 was completely substituted by the IEC 61499 compliant controllers MO'intelligence (the make of TCS-NZ). To control each conveyor section independently, it has been equipped by such a controller, a motor drive, and sufficient number of I/O interfaces, as illustrated in Fig. 10. For testing reconfiguration capabilities an extension divert loop was built, that consists of two independently controlled L-shaped sections, each including two conveyors. The left section is also equipped with a built-in diverter.

The very same CONVEYOR_SIMCTL function block type was used to control each of the sections. The conducted tests were focusing on: (i) accuracy of baggage tracking and (ii) physical reconfiguration "on the fly."

In the first test, a model of the conveyor loop was parameterized with the actual lengths and conveyor speeds as determined by measurement. Economy stop was disabled so that conveyors would continue to run without work pieces present. The system was configured and executed without any work pieces causing the conveyors to run. Work pieces were placed and removed to observe the response of the tracking system. The system models the actual position of the PE sensors, however, the visualizer still renders them at the end of the section.

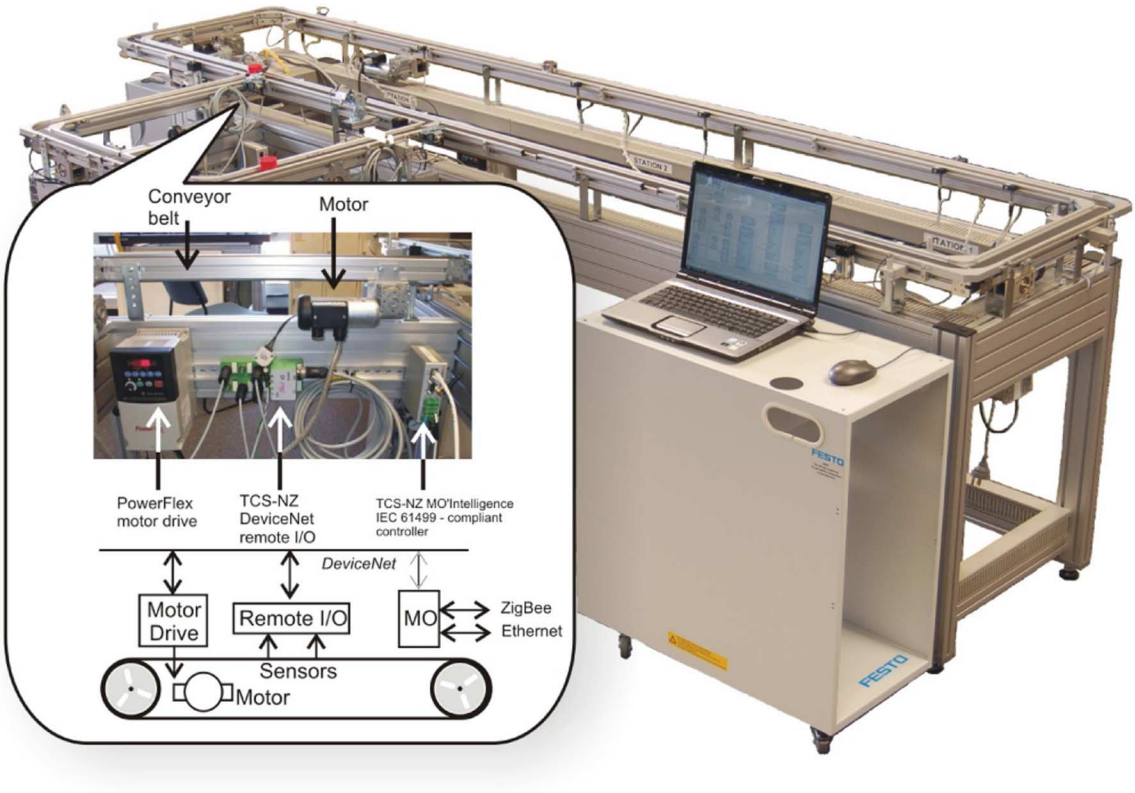


Fig. 10. Testbed for decentralized BHS control built using FESTO MPS-500 loop and two L-shaped movable conveyor sections, with embedded controller of a conveyor section.

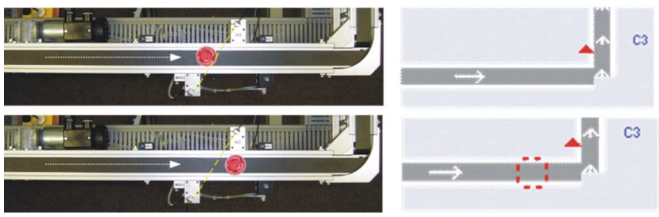


Fig. 11. Detecting a bag on the MPS 500.

Fig. 11 shows the simple case of detecting a bag as it passes through the PE. Despite the simple nature of the test, the ability of the tracking system to accurately predict the behavior of the physical system over long-time periods, and without position feedback is very pleasing. The bag tracker is an important part of the overall system because it uses the same underlying model that powers the purely simulated system as in the previous section. It is good to have some validation that the general approach will work on a real system.

In the second test, the L-shaped sections were added/removed from the system during the operation. The sections are equipped with infrared sensors which detect their docking to the main loop and to each other.

The experiments have completely proven the scalability of the developed holonic control. The rendering solution and the MVC design pattern ensured that the correct state of the systems has been displayed immediately after the physical reconfiguration without any downtime and modification of the software.

C. Distribution and Reconfiguration of Control Application

The function-block control application for the laboratory BHS testbed was tested in different hardware configurations. These experiments have proven high reconfiguration potential of the function block technology.

As illustrated in Fig. 12, the BHS control application, whose core is six interconnected instances of the function block CONVEYOR_SIMCTL was distributed across three control devices connected via Ethernet (Configuration I, upper part). In particular, a substantial subapplication, taking care of four sections in the loop was executed on the Netmaster control unit with 16 discrete inputs and 8 discrete outputs. Conveyor motors were driven by the PowerFlex motor drive devices, connected to the Netmaster via logic control signals. Two L-shaped conveyor sections were equipped with their own control devices (MO), as described in the previous section.

One reconfiguration requirement was caused by malfunctions of the Netmaster unit, which required reallocation of the conveyor loop subapplication to some other control device. The easiest remedy was to remove Netmaster from the system and connect all motor drives, sensors, and actuators of the loop via DeviceNet to the MO'intelligence controller of one of the L-shaped conveyor sections (Configuration II). The function blocks, previously residing in the Netmaster were simply moved to the MO. The whole reconfiguration has taken several minutes, after which the system was started and worked correctly. Such ease of reconfiguration would be unthinkable with the state-of-the-art PLC control.

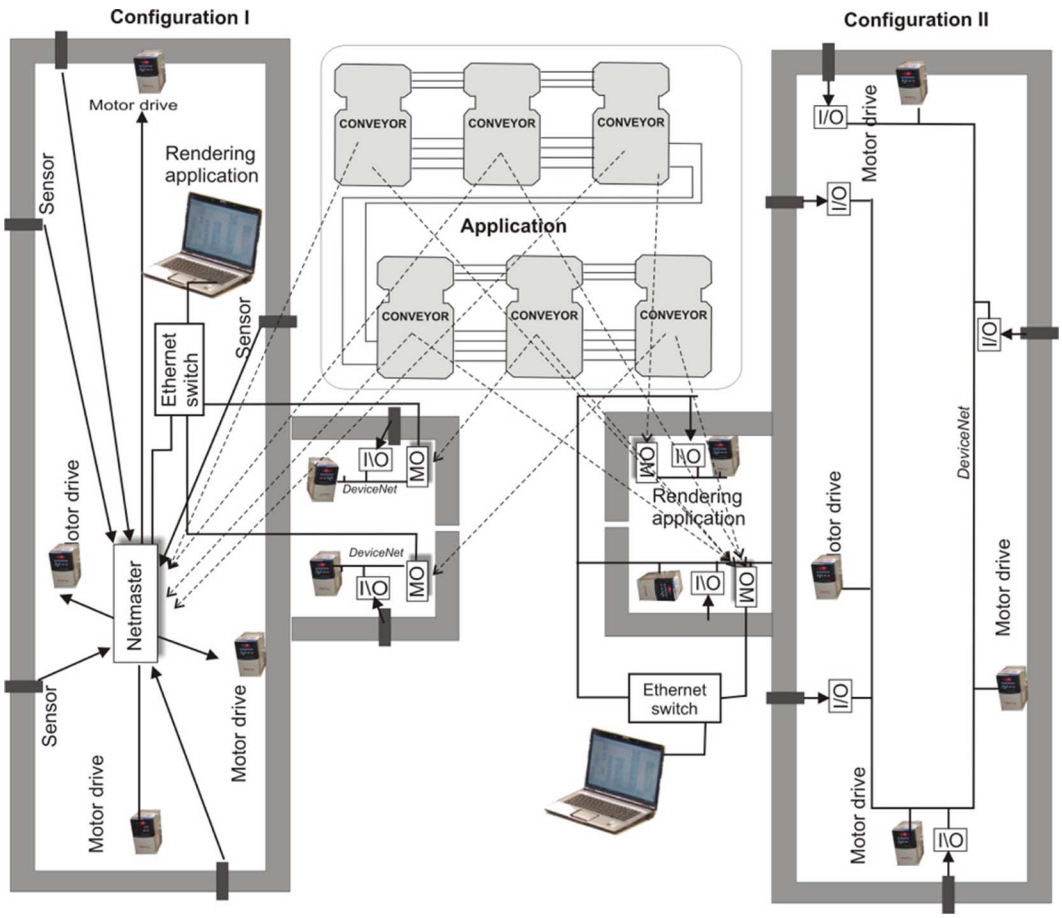


Fig. 12. The BHS application mapped on two different topologies of hardware.

VIII. Conclusion

The results, presented in this paper have advanced toward the goal of an easily reconfigurable BHS constructed of autonomous conveyor sections, each with an independent embedded control system, but collaborating to perform a traditionally centralized control application. Thus, this work paves the way to the next generation of BHS control, where each conveyor will have an embedded intelligent control device, and the whole BHS system will be able to start working immediately after assembly without extra programming.

The proposed function block-based architecture enables systematic use of the intelligent agent-based control in baggage and material handling applications. It allows benefit also from such strong features of the IEC 61499 as the ease of deployment to arbitrary topologies of hardware.

The developed system demonstrates such features of holonic control as:

- its controller is a "sum" of identical collaborative controllers of components;
- the system easily adapts to the changing environment, including changing layout, intensity of baggage flow, hardware topology and faults.

The self-configuration is not achieved at this stage, but the pathway to that within the developed architecture is quite straightforward.

The use of simulation components embedded inside a controller to achieve predictive behavior is a new use of function blocks. The bag tracking system has been demonstrated to work effectively in both simulation and on the real conveyor test system. The experimental results with the tracking system confirm the feasibility of implementing a bag tracking system using predictive simulation, not just in isolation, but in a distributed execution context. The use of the same simulation model function block for the system simulation and for the predictor demonstrates the strong encapsulation and reuse capabilities of the function block platform.

The test results demonstrate the ability of the controller to perform a variety of basic BHS tasks without centralized direction or assistance, using only those signals available on a typical conveyor.

The ability to plan bag paths using only distributed execution is mostly presented as proof of concept, but also shows possibility of being useful in reconfigurable distributed applications. The path planning system currently does not use the available metrics of conveyor and belt speed, but simply uses the first valid path discovered.

While the function block platform proved to be well suited to this project, the internal implementation of the current design relies on a significant amount of Java with the function block as a container. Importantly however, this reliance on Java is limited to only within each block—all communication between con-

veyors uses IEC 61499 standard communication blocks. The algorithms, currently implemented in Java, can be easily reimplemented in other high-level languages and encapsulated to separate function blocks, providing services to the rest of the application.

The goals of future work will be to demonstrate that it is possible for a distributed control system to dynamically self configure and achieve a basic path finding capability. More extensive case studies need to be conducted in terms of the system's scale and real-time execution and communication requirements. Adaptability of the developed control will need to be further investigated. The distributed controllers need to be integrated with the resource planning and baggage database applications.

REFERENCES

[1] K. Vickers and R. W. Chinn, "Passenger terminal baggage handling systems," *IEE Colloquium on Systems Engineering of Aerospace Projects* , Digest No. 1998/249, pp. 6/1–6/7, 1998.

[2] K. H. Hall, R. J. Staron, and A. Zoitl, "Challenges to industry adoption of the IEC 61499 standard on event-based function blocks," in *Proc. 5th IEEE Int. Con. Ind. Inform. (INDIN)*, 2007, pp. 823–828.

[3] "IEC 61499-1: Function Blocks–Part 1 Architecture," 1st ed. International Electrotechnical Commission, 2005.

[4] E. H. van Leeuwen and D. Norrie, "Holons and holarchies [intelligent manufacturing systems]," *Manuf. Eng.*, vol. 76, pp. 86–88, 1997.

[5] J. Chouinard and R. Brennan, "Software for Next Generation Automation and Control," in *Proc. 2006 IEEE Int. Conf. Ind. Inform.*, pp. 886–891.

[6] S. Bussman, N. R. Jennings, and M. Wooldridge*, Multiagent Systems for Manufacturing Control*. Heidelberg, Germany: Springer-Verlag, 2004.

[7] R. W. Brennan, "Holonic and multi-agent systems in industry," *The Knowledge Engineering Review*, vol. 16, pp. 375–381, 2001.

[8] K. Hallenborg and Y. Demazeau, "Dynamical Control in Large-Scale Material Handling Systems through Agent Technology," in *Proc. IEEE/WIC/ACM Int. Conf. Intell. Agent Technol.*, 2006, pp. 637–645.

[9] N. Hayslip, S. Sastry, and J. Gerhardt, "Networked embedded automation," *Assembly Automation*, vol. 26, pp. 235–241, 2006.

[10] FIPA, "Agent Communication Language," Jun. 2007. [Online]. Available: http://www.fipa.org/specs/fipa00003/

[11] P. Vrba, "MAST: Manufacturing agent simulation tool," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, 2003, vol. 1, pp. 282–287.

[12] V. Marik, P. Vrba, K. H. Hall, and F. P. Maturana, "Rockwell automation agents for manufacturing," in *Proc. 4th Int. Joint Conf. Autonomous Agents and Multiagent Syst.*, The Netherlands, 2005, pp. 107–113.

[13] Function Block Development Kit 2008, HOLOBLOC.

[14] C. Sünder, A. Zoitl, M. Rainbauer, and B. Favre-Bulle, "Hierarchical control modelling architecture for modular distributed automation systems," in *Proc. IEEE Int. Conf. Ind. Inform.*, 2006, pp. 12–17.

[15] C. Schwab, M. Tangermann, and L. Ferrarini, "Web based methodology for engineering and maintenance of distributed control systems: The TORERO approach," in *Proc. 3rd IEEE Int. Conf. Ind. Inform*, 2005, pp. 32–37.

[16] K. Thramboulidis, "Model-integrated mechatronics—Toward a new paradigm in the development of manufacturing systems," *IEEE Trans. Ind. Inform.*, vol. 1, pp. 54–61, 2005.

[17] G. Frey and T. Hussain, "Modeling techniques for distributed control systems based on the IEC 61499 standard—Current approaches and open problems," in *Proc. 8th Int. Workshop on Discrete Event Syst.*, 2006, pp. 176–181.

[18] R. W. Brennan, L. Ferrarini, J. Martinez Lastra, and V. Vyatkin (Eds.), "Special issue on automation objects," *Int. J. Manuf. Res.*, vol. 1, no. 4, pp. 382–518, 2006.

[19] V. V. Vyatkin, H. M. Hanisch, S. Karras, T. Pfeiffer, and V. Dubinin, "Rapid engineering and re-configuration of automation objects using formal verification," *Int. J. Manuf. Res.*, vol. 1, pp. 382–404, 2006.

[20] V. Vyatkin*, IEC 61499 Function Blocks for Embedded Control Systems Design*. Research Triangle Park, NC: Instrumentation Society of America, 2007.

[21] V. Vyatkin, "Intelligent mechatronic components: Control system engineering using an open distributed architecture," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, 2003, vol. 2, pp. 277–284.

[22] C. Sünder, A. Zoitl, T. Strasser, and B. Favre-Bulle, "Intuitive control engineering for mechatronic components in distributed automation systems based on the reference model of IEC 61499," in *Proc. 3rd IEEE Int. Conf. Ind. Inform.*, 2005, pp. 50–55.

[23] V. Vyatkin, Z. Salcic, P. Roop, and J. Fitzgerald, "Information infrastructure of intelligent machines based on IEC 61499 architecture," *IEEE Ind. Electron. Mag.*, 2007, accepted for publication.

[24] J. H. Christensen, "IEC 61499 architecture, engineering methodologies and software tools," in *Proc. 5th IFIP Int. Conf. Inf. Technol. Balanced Autom. Syst. Manuf. Services*, Cancun, Mexico, 2002, pp. 221–228.

[25] M. Hirsch, V. Vyatkin, and H. M. Hanisch, "IEC 61499 function blocks for distributed networked embedded applications," in *Proc. IEEE Int. Conf. Ind. Inform.*, 2006, pp. 670–675.

[26] J. H. Christensen, "Holonic manufacturing systems: Initial architecture and standards directions," in *Proc. 1st Euro. Wkshp. Holonic Manuf. Syst., HMS Consortium*, Hannover, Germany, Dec. 1994.

[27] M. Fletcher, "Building holonic control systems with function blocks," in *Proc. 5th Int. Symp. Autonomous Decentralized Syst.*, 2001, pp. 247–250.

[28] M. Fletcher, E. Garcia-Herreros, J. H. Christensen, S. M. Deen, and R. Mittmann, "An open architecture for holonic cooperation and autonomy," in *Proc. 11th Int. Workshop Database Expert Syst. Appl.*, 2000, pp. 224–230.

[29] L. Ferrarini and C. Veber, "Design and implementation of distributed hierarchical automation and control systems with IEC 61499," in *Proc. 3rd IEEE Int. Conf. Ind. Inform.*, 2005, pp. 74–79.

[30] G. Black and V. Vyatkin, "On practical implementation of holonic control principles in Baggage handling systems using IEC 61499," in *Holonic and Multi-Agent Systems for Manufacturing*, 2007, vol. 4659, Lecture Notes in Computer Science, pp. 314–325.

[31] T. H. Cormen*, Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001, pp. 595–601.

[32] D. G. Bertsekas and Robert*, Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

Geoff Black received the Bachelor of Engineering degree (electrical and electronic) in 1998 and the Master of Engineering degree in computer systems in 2008 from the University of Auckland, Auckland, New Zealand.

He has experience in embedded software for industrial and process control, specializing in fluid control using electronic actuation. He is now employed by Wellington Drive Technologies Limited, New Zealand, developing high-efficiency motor drives for electronically commutated motors.

Valeriy Vyatkin (SM'04) graduated with a Diploma degree in applied mathematics in 1988, received the Dr. Sci. degree in 1998, the Ph.D. degree in 1992 from Taganrog State University of Radio Engineering (TSURE), Taganrog, Russia, and the Dr. Eng. degree from the Nagoya Institute of Technology, Nagoya, Japan, in 1999.

Currently, he is Senior Lecturer with the Department of Electrical and Computer Engineering at the University of Auckland, Auckland, New Zealand. His previous faculty positions were with Martin Luther University of Halle-Wittenberg in Germany (Assistant Professor, 1999–2004), and with TSURE (Senior Lecturer, Professor, 1991–2002). He is the Head of the infoMechatronics and IndusTRial Automation lab (MITRA). Research interests are in the area of industrial informatics, including software engineering for industrial automation systems, distributed software architectures, methods of formal validation of industrial automation systems, and theoretical algorithms for improving their performance. The specific expertise area is in distributed automation and the IEC 61499 standard.