IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 17, 2019, accepted July 4, 2019, date of publication July 8, 2019, date of current version July 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2927263

A Decomposition-Based Development Method for Industrial Control Systems

JIAWEN XIONG, JU LI, JIANQI SHI, AND YANHONG HUANG
National Trusted Embedded Software Engineering Technology Research Center, East China Normal University, Shanghai 200062, China

Corresponding author: Jianqi Shi (jqshi@sei.ecnu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602178, in part by the Shanghai Science and Technology Committee Rising-Star Program under Grant 18QB1402000, and in part by the China HGJ Project under Grant 2017ZX01038102-002.

ABSTRACT Industrial control systems (ICSs), especially distributed control systems (DCSs), are usually composed of several subsystems. Each subsystem is controlled by a control unit such as a programmable logic controller (PLC) or a micro-controller and collaborates with other subsystems via the field bus, Ethernet, or other communication links. In the traditional development process, engineers program for each PLC separately and skillfully orchestrate the collaboration among subsystems, which is difficult and error-prone. The larger the scale of the ICS is, the higher the complexity of the collaboration is, and the more error-prone the development process is. In this paper, we propose a decomposition-based development method for distributed ICSs to reduce the difficulty of developing distributed ICSs whose subsystems cooperate with each other. First, we present a general event-triggered specification language named *Industrial Modeling Collaboration Language (IMCL)* for modeling ICSs; the language allows describing system functions and physical resources in one unified model. Second, we provide an approach for decomposing the complex system model into multiple fine-grained and interactive subsystem models. Specifically, under given resource constraints, we propose an automatic decomposition and collaboration algorithm based on the *IMCL* model to meet the original functional requirements. In this way, engineers can develop distributed control systems without considering the underlying complex interaction mechanisms. We present a case study to demonstrate it.

INDEX TERMS Automatic decomposition, distributed control system, industrial control system, model collaboration, model decomposition, programmable logic controller.

I. INTRODUCTION

Industrial control systems (ICSs) are the systems used for industrial process control; the systems can range from a few modular panel-mounted controllers to larger interconnected and interactive distributed control systems (DCSs). The larger ICSs usually consist of supervisory Control and Data Acquisition (SCADA) systems, DCSs, and programmable logic controllers (PLCs), etc. Such systems are extensively used in industries such as chemical processing, pulp and paper manufacture, power generation, oil and gas processing and telecommunications. As shown in Fig.1, in a industrial control application, there are massive sensors monitoring various control process variables (PVs) and transmitters transferring data to a local controller such as a PLC or an embedded micro-controller via some type of industrial field bus. By comparing these data with desired set points (SPs), the local controller derives command functions which are used to control a process through the final control elements (FCEs), such as control valves. Several local controllers usually collaborate with each other as a DCS to achieve advanced process control. Hence, an actual ICS is similar to a special distributed computer system that consists of a number of Computing Units (CUs) [1]. However, every CU has its own capability limits to handle tasks, schedule physical resources and collaborate with other CUs. For example, a controller may be not able to complete a certain calculation task within a specified time alone or some devices connected to a CU cannot be directly accessed by another CU, but are accessed through communication between these two CUs. Moreover, the more CUs and resources exist, the more complex the communication process is. Therefore, it is challenge to design a distributed

The associate editor coordinating the review of this manuscript and approving it for publication was Tao Zhang.

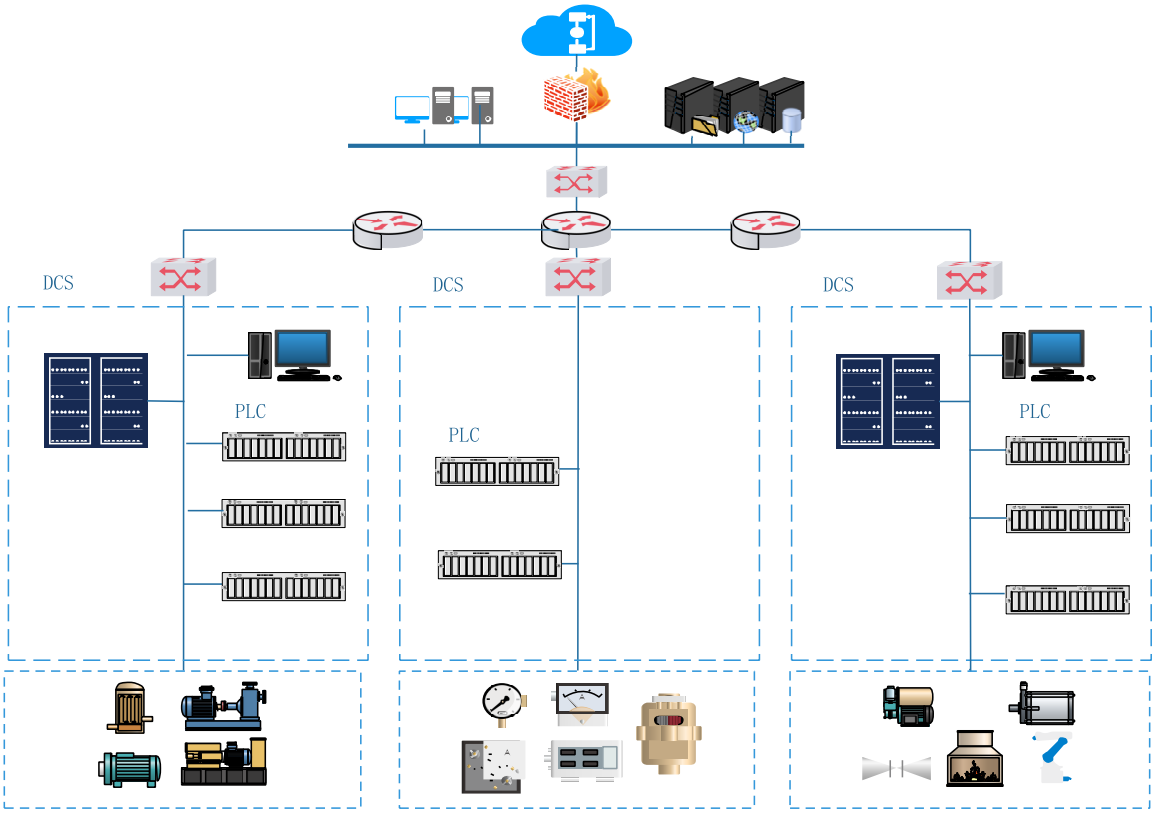


FIGURE 1. An example of an industrial control system.

system with a certain number of controllers with different capabilities and different resources. In the traditional development process, developers usually use a top-down method to design a large system in a unified manner, but separately implement the subsystems. In order to ensure the functional consistency, developers have to design an elaborate communication mechanism to schedule different controllers, and they have to perform elaborate tests or adopt formal verification techniques to ensure the functional correctness. As ICSs have become more feature-rich and more intelligent, a growing number of physical resources exist in a system and there are certainly more and more complicated communication processes, implying that the industrial control processes must be more complex than before. All of this complicates design and development of such systems and the costs increase.

To overcome these problems of increasing complexity, we propose a decomposition-based development method, which reduces the complexity of a system by decomposing the large and complex system into several subsystems that are simpler and easier to control. In this way, a complex functional model can be automatically replaced by a series of models of subsystems after decomposition. An actual ICS can be treated as a complex model. Every CU in a system is responsible for a portion of the system functions and can be considered as the corresponding model of a subsystem. If one task is too complex to be handled by one CU, it will be decomposed into multiple tasks handled by multiple CUs collaborating with each other. Moreover, since most CUs have limits in terms of physical resource scheduling, the communication used for indirect access among CUs will be automatically generated. Many researchers [2]–[4] have made great achievements regarding collaboration of ICSs. A lot of researchers have focused on the intelligent industrial ecosystem [5], [6], which handle the collection of massive data from the various devices by dynamical collaboration. However, they did not take into consideration the complex physical resources for the whole collaboration process. For example, there are three physical resources ($res1$, $res2$, $res3$) in one system and all of those resources should work together; there are two CUs (CU_1, CU_2), where CU_1 is limited to schedule $res1$ and $res2$, and CU_2 is limited to schedule $res1$ and $res3$. Neither of the two CUs can independently carry on the system functions. Therefore, it is necessary to find the best solution to ensure that the two CUs collaborate with each other to implement the system requirements.

In our research, we introduce an event-triggered language called *Industrial Modeling Collaboration Language* (*IMCL*) which is a platform-independent specification language for ICSs. With this language, we can describe the system functions and the physical resources in one unified model. Moreover, we propose the decomposition algorithms that can decompose the complex system model into a number

of subsystems that correspond to every CU with constraints of resources. This algorithm includes the program analysis of control flow and data flow and the physical resource allocation under specific resource constraints. To maintain a functional consistency between the subsystems and the original system model, we present a collaboration algorithm to automatically generate interactions among these subsystems. In addition, we present an optimization method to obtain the optimal solution for system collaboration. This approach allows us to design the whole system in a unified form and decompose the system model automatically, which facilitates the design and development for large-scale industrial DCSs. Finally, a case study is presented to illustrate the utility of the proposed approach.

A. OUTLINE

The remainder of this paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we introduce the industrial Modeling Collaboration Language (IMCL) for modeling ICSs. In Section 4, we proposes the approach for decomposition and collaboration as well as the corresponding algorithms. Section 5 provides a case study for a product inspection and sorting system. Section 6 is the conclusion.

We note that a shorter conference version (4 pages) of this paper appeared in ICECCS 2017. Our initial conference paper did not give the specific details of the IMCL and the algorithms for model decomposition and collaboration. This manuscript provides them.

II. RELATED WORK

The problem of the system decomposition and collaboration has been much discussed in the academic community during the past decades. The industry has been experimenting with different methods and new technologies to achieve better solutions for specific industrial environments [7]. In [8], it is mentioned that with the rapid advancement of information and communication technologies, particularly the Internet and Web-based technologies during the past years, various systems integration and collaboration technologies have been developed and deployed to different application domains including architecture, engineering, construction, and facilities management.

In particular, there are some approaches in the field of distributed systems and web programming that are closely related. Neubauer and Thiemann [9] proposed a multi-tier calculus and splitting transformation for constructing multi-tier applications in the sequential setting. This work is very similar to our work. However, each operation used in the sequential program needs to be initially annotated whether the operation is location independent. In our work, this kind of information can be obtained through automatic dependence analysis. In [10], a framework for conversion of sequential code into distributed program code is described. Moreover, a specific hierarchical clustering algorithm is applied to obtain the optimal distribution of the program code. The key is that this framework is used to harness the processing power of idle computers in networks by automatically distributing the user application across available resources. In other words, these computing devices are equivalent. This approach is close to automatic program parallelization. In our work, we need to consider specific resource constraints which make our computing units different. It makes our work suitable for the design of control system composed of heterogeneous devices. Reference [11] presented a high level programming language named ML5 for spatially distributed computing, which uses a type system based on modal logic to statically exclude program that used mobile resources unsafely and allows an entire distributed application to be developed and reasoned about as a unified program. However, this approach requires the corresponding compiler and runtime to support. In particular, the language is hard for industrial control system engineers to use. Conversely, *IMCL* is rather close to the program model for PLC. Serrano and Berry [12] introduced a programming language named HOP, which incorporates all the required Web-related features into a single language with a single homogeneous development and execution platform. And Links from the University of Edinburgh [13] is similar. Their goal is to make the web application development more coherent. However, in our work, on the one hand, we use a unified language to specify the application for industrial controllers; on the other hand, we focus on the automatic task assignment under some constraints.

As for specification languages, the Event-B [14] and the timed automata [15], [16] can be used to model the ICSs, and both are suitable for property verification of safety-critical systems. Especially, the Event-B has robust and commercially available tool support for specification, design, proof, and code generation. However, *IMCL* is designed as a novel language for the automatic decomposition of complex systems. After describing one complex system with the *IMCL*, developers or researchers can use the decomposition algorithm to automatically break a complex system into multiple smaller and simpler sub-systems. Importantly, these subsystems will hold the same function as the original system by collaborating based on a specific communication mechanism.

Program slicing is closely related to the decomposition approach in our work. Weiser [17] proposed the program slicing method, which slices a program based on the data-flow and control-flow analysis. Ferrante *et al.* [18], present the intermediate program representation called the program dependence graph (PDG). Larsen and Harrold [19] presented the construction of system dependence graphs for object-oriented software and described the computation of slices on their system dependence graphs. However, our language is task-oriented and event-triggered. Gauthier *et al.* [20] and Cheng [21] proved the decomposition theorems for linear programs and the approach of slicing concurrent programs, respectively. References [17]–[21] are influential regarding the basic principle of our decomposition algorithms.

In [4] and [22]–[24], many researchers have focused on multi-agent systems and have made great achievements.

Zou *et al.* [25] investigated the problem of event-triggered distributed predictive control for multi-agent systems. Olfati-Saber *et al.* [26] provided a theoretical framework for the analysis of consensus algorithms for the decomposition of multi-agent network systems with an emphasis on the role of directed information flow and robustness to changes in the network. He *et al.* [27] analyzed the decomposition method for the industrial power demand. De Gea Fernández *et al.* [6] introduced an intelligent and intuitive dual-arm robotic system for industrial human-robot collaboration. Hsieh [28] proposed a collaboration mechanism of resource donation, including unilateral resource donation and reciprocal resource donation. In our study, the collaboration algorithm is based on the premise of resource constraints.

Henzinger *et al.* presented a time-triggered language called *Giotto* [3], which provides an abstract programmer's model for the implementation of an embedded control system with hard real-time constraints. *Giotto* can be annotated with platform constraints such as task-to-host mappings and task and communication schedules. Different from *Giotto*, our research focuses on describing the system with event triggers. The NASA has released a toolkit called *AutoBayes* [29] that can decompose and converts a system model from a data analysis perspective. Valerdi *et al.* [30] proposed updates to requirements decomposition guidelines that will help generate the number of system requirements. Herberg and Lindemann [31] demonstrated the need for enhanced support for subsystem development and evaluation in large engineering systems. In [2], it is described how to transform one system into multi-subsystems using a criteria catalog and systematic requirement refinement; this article inspired our current research. In this research, we focus on the constraints of resources, which is the basis for decomposing one IMCL system. Chen *et al.* [5] introduced a framework and elaborated on research challenges about the industrial internet of things-based collaborative sensing intelligence. The authors believe that an intelligent industrial ecosystem enables the collection of massive data from the various devices that are dynamically collaborating with humans. In our work, we focus on the collaboration between CUs in an ICS with some resource constraints, which is different from the previous existing research.

III. INDUSTRIAL MODELLING COLLABORATION LANGUAGE (IMCL)

In an ICS, one task of a control unit usually consists of three steps: input, computing, and output, and each task is executed cyclically. In the input step, the control unit reads the values of the sensors. In the computing step, the control unit performs some computations such as numerical calculations and conditional judgment, etc. In the output steps, the control unit modifies the values of the variables that are mapped to some output points or control actuators to conduct certain mechanical actions by providing output signals to driver devices. Therefore, a task of a control unit can be regarded as an event trigger that waits for an event and performs certain actions when the event occurs. In particular, periodic tasks can be considered as being triggered by time events. Therefore, we proposed the industrial modeling collaboration language (IMCL) in an event-trigger format. According to IEC61131-3 (the open international standard IEC61131 for PLC), a PLC have a configuration; a configuration consists of several *resource*s (like CPUs); each *resource* consists of multiple *task*s (like processes); each *task* is bound with a *program*; each task can be executed once, on a timer or on an event. Logically, all tasks run concurrently. However, if the execution time is longer than the cycle or an event occurs during the execution of a piece of code that is managing a previous event, a new task instance is created but waits in a waiting queue to schedule. It is a task management problem. In this paper, we assume all event triggers run concurrently. Here, we will introduce the abstract syntax of the *IMCL*. The concrete syntax is provided in the Appendix A.

A. THE SYNTAX OF IMCL

The abstract syntax of the *IMCL* is defined as follows:

$$
\begin{aligned}
A_{exp} &::= val \mid x \mid (A) \mid A_0 + A_1 \mid A_0 - A_1 \mid A_0 * A_1 \\
&\quad \mid A_0/A_1 \\
B_{exp} &::= \top \mid \bot \mid A_0 = A_1 \mid A_0 \neq A_1 \mid A_0 > A_1 \mid A_0 < A_1 \\
C_{exp} &::= channel!m \mid channel?m \mid sync.n \\
E_{exp} &::= x := A \mid C \mid a \gg Dev \mid a \ll Dev \mid E_0; E_1 \\
&\quad \mid E_0 \lhd B \rhd E_1 \mid B * E \\
T_{exp} &::= trigger\ B \diamond E \mid trigger\ channel?m \diamond E
\end{aligned}
$$

A_{exp} represents arithmetic expressions where *val* is a value, x is a variable, and A_0, A_1 are instances of A_{exp}. B_{exp} represents Boolean expressions, where $\top$ and $\bot$ indicate the *true* and *false* respectively. Next, we introduce certain special expressions in the *IMCL*. C_{exp} represents the set of communication expressions. The $channel!m$ refers to sending a value m via a *channel*. The $channel?m$ refers to receiving a value m from a *channel*. The $sync.n$ refers to obtaining the synchronous data n. E_{exp} is the execution expression. The conditional choice $E_0 \lhd B \rhd E_1$ indicates that if B is true then E_0, else E_1. The $B * E$ indicates that E will iterates until B is false. Specifically, $a \gg Dev$ denotes that the system transmits the value a to a physical device *Dev*, while $Dev \ll a$ is obtaining a from *Dev*. T_{exp} represents the event-triggered expressions. The $trigger\ B \diamond E$ denotes an event E occurring when the event condition B is true. The $trigger\ channel?m \diamond E_{exp}$ indicates that the event E will begin to execute when receiving a m from the *channel*.

B. IMCL MODEL FOR INDUSTRIAL CONTROL SYSTEM

The modeling process focuses on system functions and features and includes the following two steps:

1) MODELLING THE PHYSICAL RESOURCES

Physical resources includes sensors, actuators, and other read-write devices. The representations of the physical

resources depend on the industrial environment. Considering their effects on the whole system, we describe all the resources as variables to unifying the definition of the resources.

2) MODELLING THE SYSTEM

By observing the behavior, a system integrates numerical calculations, read-write operations, and other actions. We model them as execution expressions. Multiple execution expressions in one specific order can comprise one trigger for an event marked as T.

An *IMCL* model consists of multiple concurrent event triggers, that are triggered only when the event condition is satisfied. Let $\bowtie$ be the concurrent operation of two events s.t. $T_1 \bowtie T_2$. Then, the *IMCL* model *Prog* can be defined as follows:

$$Prog = \overset{n}{\underset{i=1}{\bowtie}} T_i, \quad n \in N^+$$

Example 1. There is an example of ICS: the data collector *collects* information and saves it to the *database*. When the temperature *sensor* detects the temperature exceeds 200 degrees, the system turns on the *fan* to cool the system. The *IMCL* model of this system is shown as follows:

```
// Physical resources in system
SENSOR: sensor;
DEVICE: collector, database, fan;

program : Example() [VAR: tmp, data]
{
   TRIGGER(sensor > 200) // trigger event: T1
   {
      tmp := cool;
      tmp >> Fan;
   }

   TRIGGER(TRUE) // trigger event: T2
   {
      data << collector;
      data >> database;
   }
}
```

We abstract the physical resources in the system as variables. The T_1 and T_2 are two event triggers representing different process control functions. Therefore, the system model can be described as $Prog = T_1 \bowtie T_2$.

IV. APPROACH FOR DECOMPOSITION AND COLLABORATION

For a given *IMCL* program $Prog_{ori}$, we obtain an abstract syntax tree (AST) which contains details of its statements by parsing it using an open source grammar parser tool, ANTLR [32]. Subsequently, we obtain the corresponding control flow graph (CFG) and data flow graph (DFG) from the AST. A $CFG = \langle N, E \rangle$ is a directed graph, where N is a set of nodes, and $E \subseteq N \times N$ is the set of edges whereas *DFG* is a data-flow digraph structure. If $(n_1, n_2) \in E$, then n_2 is the immediate successor of n_1.

During the construction of the CFG, we obtain the information of each node. For each node n, we have the following three sets: *REF(n)*, *DEF(n)* and *INFL(n)*. *REF(n)* is the set of variables whose values are used at n, and *DEF(n)* is the set of variables whose values are changed at n. *INFL(n)* is the set of nodes transitively control dependent on n, and it is not empty only if n has more than one immediate successor (for example, n is a branch statement or a loop node).

There is a *post* data-dependence (DD_{post}) set of nodes for every node in the DFG, which is described as follows:

$$DD_{post}(n) = \{\, m \mid m \in N \wedge\ REF(m) \cap DEF(n) \neq \emptyset \,\}$$

The DD_{post}(n) denotes those nodes which are data-dependent on node n.

SDG: The *system dependence graph* (SDG) is a graphic representation of the system model with data dependence and control dependence; it is constructed based on the *CFG* and *DFG*:

$$SDG := CFG \oplus \bigcup_{i=1}^{N} DD_{post}(i)$$

which denotes that the SDG is a combination of the CFG and all post data-dependence.

There are five basic terms for the approach:

- **Original Model:** The original model is an *IMCL* model $Prog_{ori}$ given at the start, which is the description of an ICS.
- **Statement:** The *statement* is the minimum computational task level of the program. Therefore, the $Prog_{ori}$ can be treated as a set of statements.
- **Resource Constraints:** They describe the constraints of the physical resources that are limited available for specific CUs.
- **Decomposition Models:** These are multiple models where all of the statements in *Original Model* are decomposed into for the specified *CU*. In other words, the $Prog_{ori}$ with the resource constraints is transformed to a set of $Prog_{cu}$ for every *CU*.
- **Collaboration Model:** This is a set of the $Prog_{cu}$ interacting with each other through communication and synchronization.

Figure 2 shows an overview of the approach for decomposition and collaboration that we have implemented in our tool. For an industrial control system, we model the physical resources and the system functions in one *Original Model* ($Prog_{ori}$). Based on the SDG of the $Prog_{ori}$, we implement the decomposition and collaboration algorithms to decompose the *Original Model* into the *Collaboration Model* with the resource constraints.

A. DECOMPOSITION ALGORITHMS

We can determine the dependence of the control flow and data flow to keep the constancy of the *Original Model* and *Decomposition Model* only by analyzing SDG. Therefore, the SDG analysis is a very important part of the decomposition algorithms.

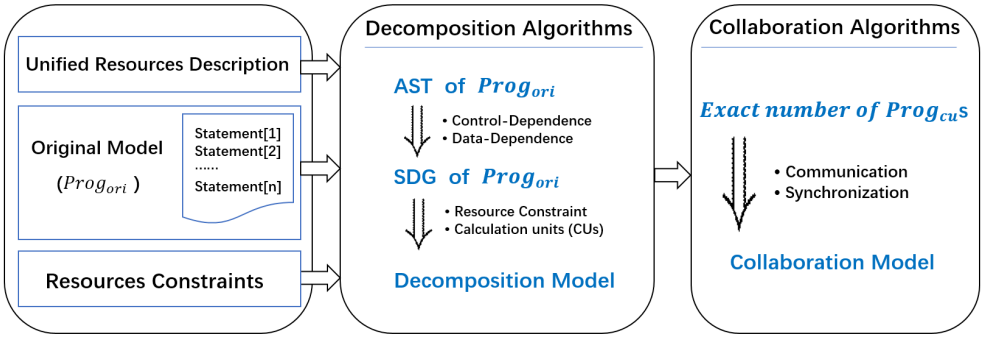


FIGURE 2. Overview of the approach for decomposition and collaboration.

1) ANALYSIS OF SDG

As mentioned before, SDG is the original model got from *AST*. We show the principal conversion procedure from *AST* to *SDG*.

In **Algorithm 1**, there are one *input AST* and one *output SDG* in this algorithm. Let V be the set of variables in a program. For each $v \in V$, Δ_v is the set of statements lately assigned before each statement is reached, and the Δ'_v is the new set after the assignment. Specifically, for two statements m and n, which have the same control-dependence statement, if m is a condition of the loop statement, then for a variable $var \in V$, the Δ'_v after reaching m is through all the relevant control statements with m rather than directly as the Δ_v of n. Therefore, it needs to recursively solve the *INFL(m)* with m before n. Example 2 is used to explain this special situation.

Example 2. Figure 3 shows the *IMCL* program $Prog_{ori}$ and the SDG obtained by Algorithm 1. The statement S_5(IF(x<2)) and S_{10}(y:=y∗2) in the figure are influenced by statement S_4(WHILE(y<100)) because the variable y in S_{10} depends on S_6 and S_8. Therefore, when calculating the relationship between the variable y and the S_{10}, it is not necessary to directly depend on the dependence of S_5 and S_{10}, but rather it is required to recursively solve the data change dependence of S_5 and its substructure. In the S_5, $\Delta_y = \{3, 10\}$, $\Delta'_y = \{3, 10\}$, but in S_{10}, $\Delta_y = \{6, 8\}$ and $\Delta'_y = \{10\}$.

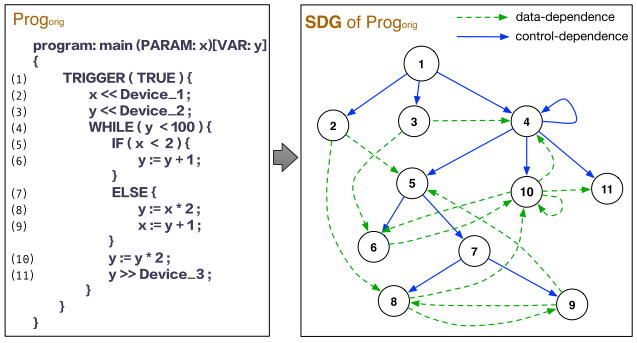


FIGURE 3. An example of a simple SDG of $Prog_{ori}$.

2) DECOMPOSITION WITH RESOURCE CONSTRAINTS

In this part, we decompose the *Original Model* into *Decomposition Model* with resource constraints. DD_{pre}, the *pre* data-dependence for each statement, can be obtained from the corresponding DD_{post}. The definition is as follows:

$$DD_{pre}(n) = \{\, m \mid m \in N \ \wedge \ n \in DD_{post}(m) \,\}$$

where m is a statement that is data-dependent on the statement in $DD_{pre}(m)$.

The *Algorithm* 2 presents the principal procedures of the decomposition algorithm. It includes two **inputs**:

i. SDG: The system dependence graph of *Original Model* where every node in the tree is a statement with *REF(node)*, *DEF(node)*, and *INFL(node)*.
ii. resConst: A mapping of the resource constraints to *CUs*.

The **output** of this algorithm is *decompModel*, which is a *Decomposition Model* with all statements labeled by the specified *CUs*.

The core of this **Algorithm 2** comprises three steps:

i. Firstly, if one statement is associated with one resource, we label the statement with the *CU* corresponding to the constraint of this resource.
ii. Secondly, for every statement n labeled with *CU*, we will label the same *CU* for every statement in $DD_{pre}(n)$ that has not been labeled; this ensures that the influence on the decomposition for every resource is as fair as possible.
iii. After the two preceding steps, there are still some unlabeled statements, then we label them with the *CU* in the forward control-dependence statement.
This indicates that if there are statements n and m in *INFL(n)*, then the m will be labeled with *CU* in n.

The summary of algorithm 2 is that it requires $\mathcal{O}(|N||R| + 2|N||Dep|)$ time, and $\mathcal{O}(|N| + |R| + |DD_{pre}| + |INFL|)$ space.

B. COLLABORATION ALGORITHM

In this section, we describe the collaboration algorithm. The vital part of this algorithm is that the optimized solution for collaboration can be obtained after an evaluation. The key to ensuring the reliability of *Original Model* and *Collaboration*

Algorithm 1 Converting *AST* to *Generated SDG*

Input: (1) Abstract syntax tree of *Original Model*, **AST**;

Output: (2) System dependence graph, **SDG**;

CFG $\leftarrow_{ANTLR}$ AST;
for *all var* $\in$ *V* **do**
  $Set\langle Int\rangle\ \Delta_v := \emptyset$;
  Int index := 0;
  analyzeDD(Δ_v, index, *var*);
end
$SDG := CFG \oplus \bigcup_{i=1}^{N} DD_{post}(\mathrm{i})$;
function `analyzeDD` ($Set\langle Int\rangle\ \Delta_v$, *Int index*, *Var var*)
  for *all m* $\in$ INFL(index) **do**
    if *Exp(m)* $\in$ *{'IF', 'WHILE'}* **then**
      $Set\langle Int\rangle\ \Delta'_v := \emptyset$;
      for *all j* $\in$ *INFL(m)* **do**
        for *all i* $\in \Delta_v$ **do**
          if *REF(i)* $\cap$ *{var}* $\neq \emptyset$ **then**
            $DD_{post}(\mathrm{i}) := DD_{post}(\mathrm{i}) \cup \{\mathrm{j}\}$;
          end
        end
        $\Delta'_v := \Delta'_v \cup$ analyzeDD(Δ_v, j, *var*);
      end
      $\Delta_v := \Delta'_v$;
    else
      $Set\langle Int\rangle\ \Delta'_v := \emptyset$;
      for *all i* $\in \Delta_v$ **do**
        if *REF(m)* $\cap$ *{var}* $\neq \emptyset$ **then**
          $DD_{post}(\mathrm{i}) := DD_{post}(\mathrm{i}) \cup \{\mathrm{m}\}$;
        end
        if *DEF(m)* $\cap$ *{var}* $\neq \emptyset$ **then**
          $\Delta'_v := \Delta'_v \cup \{\mathrm{m}\}$;
        else
          $\Delta'_v := \Delta'_v \cup \{\mathrm{i}\}$;
        end
      end
      $\Delta_v := \Delta'_v$;
    end
  end
  return Δ_v;

Algorithm 2 Decompose Model With the Resource Constraints

Input: (1) The **SDG** of system;
(2) The *Resource Constraint*, **resConst**;

Output: (1) The *decompModel* that all statements are labeled with specified *CU*;

$Set\langle Resource\rangle$ R = {res | res $\in$ resMap.keySet() };
for *all n* $\in$ *N* **do**
  decompModel.set(n, $\varnothing$);
  for *all res* $\in$ *R* **do**
    if *res* $\in$ REF(n) || *res* $\in$ DEF(n) **then**
      decompModel.set(n, resmap.get(res));
    end
  end
end
for *all n* $\in$ *N* **do**
  expandLabels(n, $DD_{pre}(n)$);
end
for *all n* $\in$ *N* **do**
  expandLabels(n, *INFL(n)*);
end
function `expandLabels` (*Int current*, $Set\langle Int\rangle$ *Dep*)
  for *all m* $\in$ *Dep* **do**
    if *decompModel.get(m)* $= \varnothing$ **then**
      decompModel.set(m, resmap.get(m));
    end
  end

Model is the communication during collaboration, which is based on the SDG analysis.

1) COMMUNICATION OF COLLABORATION

We abstract the communication protocols between multiple models. The *IMCL* unifies the communication and data synchronous among multiple models. There are three specific communication methods as follows:

- **CHANNEL.CD!x** and **CHANNEL.CD?x**: the symbol *CHANNEL.CD* is used for the control message and applied to the communication with control-dependence in multiple CUs or multiple internal event triggers in a single CU. *CHANNEL.CD!x* refers to transmitting a control message *x* via the channel; *CHANNEL.CD?x* denotes receiving the control message *x* via the channel.
- **CHANNEL.DD!x:data** and **CHANNEL.DD?x:data**: *CHANNEL.DD* is used for data message and is applied to the communication with data-dependence in multiple CUs or multiple internal triggers in a single CU. *CHANNEL.DD!x:data* refers to transmitting a data message *x* with *data* via the channel; *CHANNEL.DD?x:data* denotes receiving message x to data via the channel.
- **SYNC.DATA:data**: the *SYNC.DATA* is used to synchronize the data in different event triggers. Both *CHANEL.DD* and *SYNC.DATA* can be used to transfer data; however, unlike *CHANEL.DD*, *SYNC.DATA* does not have any dependence on the *data* in the *Original Model*.

2) COLLABORATION OF DECOMPOSED MODELS

We will review some definitions. An *Original Model* is an IMCL program consisting of multiple event triggers. The program $Prog_{cu}$ of each *CU* is obtained based on the *decompModel* after the process of decomposition.

The **Algorithm 3** presents the main procedure of the collaboration algorithm. It has three **inputs**:

Algorithm 3 Collaboration of models.

Input: (1) Original model $Prog_{ori}$ and *decompModel*; (2)$Prog_{cu}$ for every CU.

Output: (1) $Prog'_{cu}$ with *CHANNEL.CD,CHANNEL.DD and SYND.DATA* ;

```
for ∀ n, m ∈ N, n ≠ m do
    if m ∈ INFL(n) then
        if T_cu^m ≠ T_cu^n then
            CHANNEL.CD ⟨ Prog_cu(n), Prog_cu(m) ⟩ ;
        end
    end
    if m ∈ DD_post(n) then
        if decompModel.get(m) = decompModel.get(n) then
            if T_cu^m ≠ T_cu^n then
                collaborateStatements (n, m);
            end
        else
            collaborateStatements (n, m);
        end
    end
end
function collaborateStatements (Statement n, Statement m)
    if T_ori^m = T_ori^n then
        if Exp(n) ∈ {'IF', 'WHILE'} then
            SYNC.DATA ⟨ Prog_cu(n), Prog_cu(m) ⟩ ;
        else
            CHANNEL.DD ⟨ Prog_cu(n), Prog_cu(m) ⟩ ;
        end
    else
        SYNC.DATA ⟨ Prog_cu(n), Prog_cu(m) ⟩ ;
    end
```

i. $Prog_{ori}$: An *Original model* is a set of event-triggers or a set of statements; T^i_{ori} is one event-trigger in $Prog_{ori}$ that contains the statement i.
ii. decompModel: this is obtained from the Algorithm 2;
iii. The exact number of $Prog_{cu}$: the $Prog_{cu}(n)$ denotes the statement n in a $Prog_{cu}$; T^i_{cu} is the event trigger in one $Prog_{cu}$ where the statement i is.

The **output** is the actual number of $Prog'_{cu}s$ with CHANNEL.CD, CHANNEL.DD, and SYND.DATA.

First, we should consider the control-dependence of the statements in CUs. For any two different statements n and m where m is control-dependent on n in $Prog_{ori}$, if the two different statements are in different event triggers of $Prog_{cu}$s, we should implement the *CHANNEL.CD* to collaborate the two statements. The *CHANNEL.CD*$\langle n, m\rangle$ indicates that the CU containing n sends a *CHANNEL.CD!x* message after executing the n, and the CU containing m will not execute the m until receiving the message *CHANNEL.CD?x*.

Second, we will consider the data-dependence of the statements in the CUs. For any two different statements n and m where the m is data-dependent on n in $Prog_{ori}$, if both n and m are in the same CU but in different event triggers, then we handle the collaboration like *collaborateStatements*. The *collaborateStatements* describes the process in which if the n is a structure containing more than one immediate successor, then we should implement the CHANNEL.DD to collaborate the two statements. Otherwise, we implement the *SYNC.DATA* to collaborate these statements. If both n and m are in different CUs, we will handle the collaboration in the same manner as in the function *collaborateStatements*. The *CHANNEL.DD* $\langle n, m\rangle$ indicates that the CU containing n sends a *CHANNEL.CD!x:data* message with data after executing the n, and the CU containing m will not execute the m until it receives the message *CHANNEL.CD?x:data* with data. The *SYNC.DATA*$\langle n, m\rangle$ indicates that the CU containing m synchronize the variable rewrite by n at any time.

The summary of algorithm 3 is that it requires $\mathcal{O}(|N|^2|T|)$ time, and $\mathcal{O}(2|N| + |T_{cu}| + |T_{ori}|)$ space.

C. OPTIMIZATION OF COLLABORATION MODEL

During the collaboration process, it is important to analyze and evaluate some of the characteristics of the collaboration models to get the optimal solution. For a collaboration model, it is necessary and efficient to reduce the number of communications between the CUs as much as possible. Because some resources can be allocated to different CUs, there is a need to evaluate the different results to determine the best collaboration model.

1) RESOURCE ALLOCATION

Some resources in the system model can be scheduled by a number of different CUs. Therefore, different allocations of those resources will cause different resource constraints and will contribute to different collaboration models. For example, there are four resources ($res1$, $res2$, $res3$, $res4$) and the relationship of the CUs mapping resources is:

$$A_{cu} : \{\ res1,\ res2,\ res4\ \}$$
$$B_{cu} : \{\ res1,\ res3,\ res4\ \}$$

They can be combined into the following four different resource constraints because the $res1$ and $res4$ can be allocated to either A_{cu} or B_{cu}:

Allocation 1 :
$$\{res1 : A_{cu},\ res2 : A_{cu},\ res3 : B_{cu},\ res4 : A_{cu}\}$$
Allocation 2 :
$$\{res1 : A_{cu},\ res2 : A_{cu},\ res3 : B_{cu},\ res4 : B_{cu}\}$$
Allocation 3 :
$$\{res1 : B_{cu},\ res2 : A_{cu},\ res3 : B_{cu},\ res4 : A_{cu}\}$$
Allocation 4 :
$$\{res1 : B_{cu},\ res2 : A_{cu},\ res3 : B_{cu},\ res4 : B_{cu}\}$$

2) OPTIMIZATION OF THE COLLABORATION MODEL

In this study, we suggest an evaluation strategy to help researchers and developers to find the relatively optimal collaboration model. The evaluation is presented in the following equation:

$$Eval(Sol_i) = \frac{CD_i - CD_{min}}{CD_{max} - CD_{min}} + \frac{DD_i - DD_{min}}{DD_{max} - DD_{min}} + \frac{SYNC_i - SYNC_{min}}{SYNC_{max} - SYNC_{min}}$$

The proposed evaluation equation is based on the normalization. We use the equation to compare the different effects of the results. It includes three parts: *control dependence*, *data dependence* and *synchronization*, and we define CD, DD and SYNC as the number of CHANNEL.CD, CHANNEL.DD, and SYNC.DATA, respectively, in one collaboration solution.

For the example of the part of CHANNEL.CD, the minimum data of the serial CD_i is identified as CD_{min} and the maximum one is identified as CD_{max} by the result of all solution statistics. Furthermore, it is a common way to normalize the data by the construct of normalization to map every value of CD_i to a fixed range between 0 and 1 for scaling purposes.

The expressions of CHANNEL.DD and SYNC.DATA expression are similar to the CHANNEL.CD except that the object of the control dependence is replaced by the data dependence or synchronization. The last step is to sum the value of the three parts, and the smaller the value of Eval(Sol_i) is, the better the solution is. Therefore, the optimization function can be written as follows:

$$\min_{i \in S} Eval(Sol_i)$$

where S is the set of all possible solutions. This function can help us find the solution with relatively minimum dependencies. Here, the solution S_j is optimal if and only if $Eval(Sol_j) == \min_{i \in S} Eval(Sol_i)$.

In practice, however, different evaluation strategies should be selected for different design goals. Moreover, there may be some constraints would make some solutions infeasible. For example, under the limitations of existing communication technologies, some real-time constraints may not be satisfied after decomposition. To overcome this kind of problems, some auxiliary constraints have to be supplemented into the decomposition and the collaboration algorithm.

V. CASE STUDY

An example of a real industrial *product inspection and sorting system* is used to demonstrate the approach.

A. PRODUCT INSPECTION AND SORTING SYSTEM

The product inspection and sorting system is used for product quality inspection and automatic sorting. A path delivery facility delivers products to two conveyors in turns. Both the two conveyors have a *read device* and a *write device*. The read device obtains some information from a product to check the quality of the product, and then the write device prints a detection result mark on the product. Then the sorting section scans the detecting mark on the product and decides to sort the product into the qualified area or the disqualified area.

B. IMPLEMENT OF SYSTEM MODEL

As introduced above, we will describe the modeling of the system using *IMCL*. The implementation of the system model includes three parts:

(1) Unifying the resources. *SENSOR* and *DEVICE* are the two types of resources that the system uses to get physical information and to control.

$$SENSOR :\{ pathSensor,\ sensor1,\ sensor2,\ sortSensor \};$$
$$DEVICE :\{ PATHSET,\ SREAD1,\ SWRITE1,\ SREAD2,\ SWRITE2,\ SCANNER,\ SORTSET \};$$

(2) Modeling the system. Figure 4 shows the model of the *product inspection and sorting system*. The model contains five event triggers. The first one is triggered when *pathSensor* is true, and *PATHSET* will take a product to the two conveyors. The second one is triggered when the *sensor1* is true; Then the *SREAD1* reads the information from the product to check whether it is qualified or not, and then the *SWRITE1* marks the checking result on the product and so does the third event trigger. The fourth one indicates that when the *pathSensor* is true, the sorting device will determine the destination of the product based on the mark on the product scanned by the *SCANNER*. The fifth one indicates that the system checks the system after the *picked* becomes true.

(3) Defining resource constraints. The resources constraints describe the abilities of every *CU*. For example, the computing unit B_{cu} can only control two resources *pathSensor* and *PATHSET* in this system model.

$$constraint : A_{cu} \{SCANNER,\ sortSensor,\ SWRITE1,\ SWRITE2\};$$
$$constraint : B_{cu} \{pathSensor,\ PATHSET\};$$
$$constraint : C_{cu} \{sensor1,\ sensor2,\ SREAD2,\ PATHSET\};$$
$$constraint : D_{cu} \{SREAD1,\ SWRITE1,\ SREAD2,\ SORTSET\};$$

C. DECOMPOSITION AND COLLABORATION

Based on the implementation of the model, a collaboration model is obtained after decomposing the model with the resource constraints. The whole process consists of the resource allocation, decomposition of the model, and the collaboration with evaluation.

(1) Resource allocation: *SREAD2*, *SWRITE1*, and *PATHSET* are all resource constraints that they can be used by more than one *CU*; eight solutions satisfying these resource constraints are generated, and are listed in Table 1.

(2) Optimization of Decomposition and Collaboration: There are eight results shown in Table 2 corresponding to the eight solutions in Table 1.

```
      program : sorting (PARAM: cycles, countQuality, countUnQuality)
(Entry) [VAR: cycles, countQuality, countUnQuality, info1, info2, mark1, mark2, mark, picked]
      {
 (1)    TRIGGER (pathSensor == TRUE) {
 (2)      cycles := cycles + 1;
 (3)      IF (cycles % 2 == 1) {
 (4)        PATHSET << 'path1';
          }
 (5)      ELSIF (cycles % 2 == 0) {
 (6)        PATHSET <<  'path2';
          }
        }
 (7)    TRIGGER (sensor1 == TRUE) {
 (8)      info1 << SREAD1;
 (9)      mark1 := checkProduct(info1);
(10)      mark1 >> SWRITE1;
        }
(11)    TRIGGER (sensor2 == TRUE) {
(12)      info2 << SREAD2;
(13)      mark2 := checkProduct(info2);
(14)      mark2 >> SWRITE2;
        }
(15)    TRIGGER (sortSensor == TRUE) {
(16)      mark << SCANNER;
(17)      IF (mark == TRUE) {
(18)        SORTSET << 'Quality';
(19)        countQuality := countQuality + 1;
          }
(20)      ELSE {
(21)        SORTSET << 'UnQuality';
(22)        countUnQuality := countUnQuality + 1;
          }
(23)      picked := TRUE;
        }

(24)    TRIGGER (picked == TRUE) {
(25)      IF(countUnQuality / countQuality > 0.01) {
(26)        STOP;
          }
(27)      picked := FALSE;
        }
      }
```

FIGURE 4. Modeling the product inspection and sorting using IMCL.

TABLE 1. Solutions that CUs are allocated with resources.

Resources	CUs allocated with resources (CU_*)							
	Sol 1	**Sol 2**	**Sol 3**	**Sol 4**	**Sol 5**	**Sol 6**	**Sol 7**	**Sol 8**
sortSensor	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}
*sensor*2	C_{cu}	C_{cu}	C_{cu}	C_{cu}	C_{cu}	C_{cu}	C_{cu}	C_{cu}
PATHSET	B_{cu}	C_{cu}	B_{cu}	C_{cu}	B_{cu}	C_{cu}	B_{cu}	C_{cu}
*SWRITE*1	A_{cu}	A_{cu}	D_{cu}	D_{cu}	A_{cu}	A_{cu}	D_{cu}	D_{cu}
*SWRITE*2	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}
pathSensor	B_{cu}	B_{cu}	B_{cu}	B_{cu}	B_{cu}	B_{cu}	B_{cu}	B_{cu}
SORTSET	D_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}
SCANNER	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}	A_{cu}
*sensor*1	C_{cu}	C_{cu}	C_{cu}	C_{cu}	C_{cu}	C_{cu}	C_{cu}	C_{cu}
*SREAD*1	D_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}
*SREAD*2	C_{cu}	C_{cu}	C_{cu}	C_{cu}	D_{cu}	D_{cu}	D_{cu}	D_{cu}

As shown in Fig.4, there are 27 statements in the model and every statement is the minimum computational task we want to decompose. Table 1 shows the eight solutions to decompose the model. In every solution, each *CU* has a set of statements, which means that the *CU* can take the minimum computational tasks in the original model. For example, in *Sol 1*, the C_{cu} decomposed with the set {7, 11, 12} means that the C_{cu} are labeled to statement 7, 11 and 12 in the sorting system model. The *Collaborations* in this table have three sub-parts: the *CD* shows the number of control dependencies in the collaborative system; the *DD* represents the number of data dependence in the collaborative system; the *SYNC* is the number of data synchronizations in the collaborative system. The *Evals* is the evaluation of the collaboration for every solution and it is the valuation standard of various solutions.

The results in Table 2 indicates that the *Sol* 3 is the optimal solution to decompose and collaborate for the model. The collaboration of four CUs is shown in Fig.5. A comparison of the original model with the decomposition model indicates that:

- The first event trigger of the original model is assigned to B_{cu}.
- The second event trigger is decomposed and implemented by collaboration between C_{cu} and D_{cu}.
- The third event trigger is distributed on C_{cu} and A_{cu}.
- The forth event trigger is distributed on A_{cu} and D_{cu}.
- The fifth event trigger is assigned to B_{cu}.

In summary, based on the result of the dependence analysis, some triggers are not modified and just transferred from the original computing unit(CU) to the new CUs. As for the

TABLE 2. The results of our algorithms for different evaluations.

	Decompositions				Collaborations			Evals
	A_{cu}	B_{cu}	C_{cu}	D_{cu}	*CD*	*DD*	*SYNC*	
Sol 1	{ 9,10,13,14,15,16, 17,19,20,22,23 }	{1,2,3,4,5,6, 24,25,26,27}	{7,11,12}	{8,18,21}	7	2	3	1.833
Sol 2	{ 9,10,13,14,15,16, 17,19,20,22,23 }	{1,2,3,5,24, 25,26,27}	{4,6,7, 11,12}	{8,18,21}	11	2	2	2.167
Sol 3	{ 13,14,15,16,17, 19,20,22,23 }	{1,2,3,4,5,6, 24,25,26,27}	{7,11,12}	{8,9,10, 18,21}	**6**	**1**	**2**	**0.333**
Sol 4	{ 13,14,15,16, 17,19,20,22,23 }	{1,2,3,5,24, 25,26,27}	{4,6,7, 11,12}	{8,9,10, 18,21}	10	1	2	1.0
Sol 5	{ 1,2,3,4,5,6, 24,25,26,27 }	{1,2,3,4,5,6, 24,25,26,27}	{7,11}	{8,12,18,21}	8	2	2	1.667
Sol 6	{ 9,10,13,14,15,16, 17,19,20,22,23}	{1,2,3,5,24, 25,26,27}	{4,6,7,11}	{8,12,18,21}	12	2	1	2.0
Sol 7	{ 13,14,15,16,17, 19,20,22,23 }	{1,2,3,4,5,6, 24,25,26,27}	{7,11}	{8,9,10, 12,18,21}	7	1	2	1.167
Sol 8	{13,14,15,16,17, 19,20,22,23 }	{1,2,3,5,24, 25,26,27}	{4,6,7,11}	{8,9,10, 12,18,21}	11	1	3	1.5

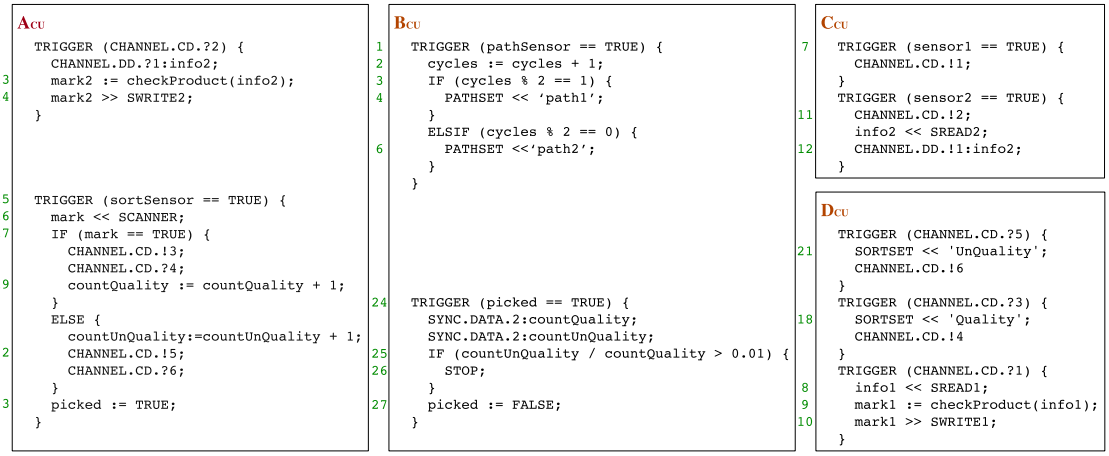


FIGURE 5. The collaboration of A_{cu}, B_{cu}, C_{cu} and D_{cu}.

remaining event triggers, which have to execute on multiple CUs and collaborate through communication, the only thing that happens is that these event triggers are split up and then reconnected with communication operations across CUs. The data flow and the control flow of the original monolithic program do not change, which means the execution semantic of these event triggers do not change on a logic level.

The implementation of this case study can be cloned from our github repository.[1]

VI. CONCLUSION

Due to the increasing complexity of ICSs or specific resources limits, an ICS is usually divided into several subsystems that collaborate with each other. Although engineers usually use top-down approaches to design these systems and break the original system into multiple subsystems, the collaborative processes among these subsystems are still designed and implemented manually in the traditional development process, which is difficult and error-prone. To overcome this problem, we propose a model language called *IMCL* to model the ICSs by taking into consideration the unifying resources and system functions. Then we develop the decomposition and collaboration algorithms based on the SDG analysis, and use the algorithms to decompose the complex system model with given resource constraints into multiple collaborating submodels of the control units efficiently and reliably. Moreover, using an optimization strategy, we obtain the optimal collaboration model of the algorithms by comparing the collaboration solutions. Therefore, our proposed approach can effectively improve the flexibility and practicality of the development for ICSs; it allows developers to avoid making mistakes and saves more time and energy for the subsequent implementation.

In the future, we plan to integrate additional constraints into the decomposition algorithms such as the cost of specific communications.

[1] https://github.com/JiawenXiong/ModelDecomposer

APPENDIX A
THE SYNTAX OF IMCL

```
// lexer rules:

ID  : ('a'..'z' | 'A'..'Z' | '_')
      ('a'..'z' | 'A'..'Z' | '0'..'9')*;

VALUE : [-]?([0-9]+[.])?[0-9]+ ;

BOOLEAN : 'TRUE' | 'FALSE';

STRING : '\'' (~('\'') )* '\'';

CALC : '+' | '-' | '*' | '/' | '%';

RELATION : '>' | '<' | '<='
         | '>=' | '==' | '!=';

INVOKE : '<<' | '>>';

WS : [ \t\r\n]* -> skip;

% // parser rules

languageIMCL :
    (codeBody)*
    ;

// codeBody
codeBody :
    processDefine
    | resourceDefine
    | constraintDefine
    ;

resourceDefine :
    ('SENSOR'|'DEVICE') ':' varAtom
    (',' varAtom)* ';'
    ;

processDefine :
    ('program'|'function') ':'
    ID '(' varDefine?  ')'
    '[' varDefine ? ']'
    '{' codeBlock*'}'
    ;

// codeBlock
codeBlock :
    assignDefine
    | ifDefine
    | whileDefine
    | triggerDefine
    | channelDefine
    ;

channelDefine :
    channel ';';

/** subDefine */
// trigger
triggerDefine :
    'TRIGGER' '('conditionExpr')'
    '{'codeBlock+ '}'
    ;

// whileExpr
whileDefine :
    'WHILE' '('conditionExpr')'
    '{'codeBlock+ '}'
    ;

// ifExpr
ifDefine :
    'IF' '('conditionExpr ')'
    '{'codeBlock*'}' (elsifDefine)*
    elseDefine*
    ;

elsifDefine :
    'ELSIF' '('conditionExpr ')'
    '{'codeBlock*'}'
    ;
elseDefine :
    'ELSE''{'codeBlock*'}'
    ;

assignDefine :
    varAtom ':=' conditionExpr ';'
    # assignVariable
    | (varAtom (',' varAtom)*
    ':=')? functionExpr ';'
    # assignFunction
    | (varAtom|valueAtom)
    INVOKE (varAtom|valueAtom)';'
    # assignInvoke
    | 'RETURN' ':' (varAtom | valueAtom)
    (',' (varAtom | valueAtom))* ';'
    # assignReturn
    | 'STOP' ';'
    # assignStop
    ;

/** subExpr */
conditionExpr :
    (varAtom | valueAtom )
    ( op=(CALC|RELATION)
    (varAtom| valueAtom) )*
    | channel
    ;
```

```
varDefine :
    ('PARAM' | 'VAR') ':'
    varAtom (',' varAtom)*
    ;

functionExpr :
    ID '(' (( valueAtom|varAtom)
    (',' (valueAtom | varAtom) )* )? ')'
    ;

varAtom :
    ID
    ;

valueAtom :
     VALUE
    | BOOLEAN
    | STRING
    ;

// Channel
channel :
    'CHANNEL.DD.' ('!'|'?')
    (ID|VALUE)  ':' varAtom (';')?
    | 'CHANNEL.CD.' ('!'|'?')
    (ID|VALUE) (';')?
    | 'DATA.SYNC.' ID ':'
    varAtom (';')?
    ;

// resources constraint
constraintDefine :
    'constraint' ':' varAtom '{'
    varAtom (',' varAtom)* '}' ';'
    ;
```

REFERENCES

[1] J. S. Gero and M. A. Rosenman, "A conceptual framework for knowledge-based design research at Sydney University's design computing unit," *Artif. Intell. Eng.*, vol. 5, no. 2, pp. 65–77, Apr. 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/095418109090003M

[2] B. Penzenstadler, "Desyre: Decomposition of systems and their requirements: Transition from system to subsystem using a criteria catalogue and systematic requirements refinement," Ph.D. dissertation, Fac. Comput. Sci., Technical Univ. Munich, Munich, Germany, 2010. [Online]. Available: http://mediatum.ub.tum.de/node?id=999357

[3] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proc. IEEE*, vol. 91, no. 1, pp. 84–99, Jan. 2003.

[4] M. Y. Damavandi, "Modeling a cooperation environment for flexibility enhancement in smart multi-energy industrial systems," Ph.D. dissertation, Univ. Beira Interior, Covilhã, Portugal, 2016. [Online]. Available: https://ubibliorum.ubi.pt/bitstream/10400.6/4366/1/Thesis-Maziar.pdf

[5] Y. Chen, G. M. Lee, L. Shu, and N. Crespi, "Industrial Internet of Things-based collaborative sensing intelligence: Framework and research challenges," *Sensors*, vol. 16, no. 2, p. 215, Feb. 2016. doi: 10.3390/s16020215.

[6] J. de Gea Fernández, D. Mronga, M. Günther, M. Wirkus, M. Schröer, S. Stiene, E. Kirchner, V. Bargsten, T. Bänziger, J. Teiwes, T. Krüger, and F. Kirchner, "iMRK: Demonstrator for intelligent and intuitive human–robot collaboration in industrial manufacturing," *KI - Künstliche Intelligenz*, vol. 31, no. 2, pp. 203–207, Jan. 2017. doi: 10.1007/s13218-016-0481-5.

[7] J. Korhonen, "Four ecosystem principles for an industrial ecosystem," *J. Cleaner Prod.*, vol. 9, no. 3, pp. 253–259, Jun. 2001.

[8] W. Shen, Q. Hao, H. Mak, J. Neelamkavil, H. Xie, J. Dickinson, R. Thomas, A. Pardasani, and H. Xue, "Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review," *Adv. Eng. Informat.*, vol. 24, no. 2, pp. 196–207, Apr. 2010.

[9] M. Neubauer and P. Thiemann, "From sequential programs to multi-tier applications by program transformation," in *Proc. 32nd ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang.*, Long Beach, CA, USA, Jan. 2005, pp. 221–232. doi: 10.1145/1040305.1040324.

[10] S. Parsa and V. Khalilpoor, "Automatic distribution of sequential code using JavaSymphony middleware," in *Proc. Int. Conf. Current Trends Theory Pract. Comput. Sci.*, 2006, pp. 440–450. doi: 10.1007/11611257_42.

[11] T. Murphy VII, K. Crary, and R. Harper, "Type-safe distributed programming with ML_5," in *Proc. Int. Symp. Trustworthy Global Comput.*, 2007, pp. 108–123. doi: 10.1007/978-3-540-78663-4_9.

[12] M. Serrano and G. Berry, "Multitier programming in Hop," *ACM Queue*, vol. 10, no. 7, pp. 53–59, Aug. 2012. doi: 10.1145/2330087.2330089.

[13] E. Cooper, S. Lindley, P. Wadler, and J. Yallop, "Links: Web programming without tiers," in *Proc. Int. Symp. Formal Methods Compon. Objects*, 2006, pp. 266–296. [Online]. Available: https://doi.org/10.1007/978-3-540-74792-5_12

[14] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[15] R. Alur and D. L. Dill, "A theory of timed automata," *Theory Comput. Scince*, vol. 126, no. 2, pp. 183–235, 1994.

[16] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets, Advances in Petri Nets* (Lecture Notes in Computer Science), vol. 3098, J. Desel, W. Reisig, and G. Rozenberg, Eds. Berlin, Germany: Springer, 2003, pp. 87–124. doi: 10.1007/978-3-540-27755-2_3.

[17] M. Weiser, "Program slicing," in *Proc. 5th Int. Conf. Softw. Eng. (ICSE)*. San Diego, CA, USA: IEEE Press, 1981, pp. 439–449.

[18] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, pp. 319–349, 1987. doi: 10.1145/24039.24041.

[19] L. Larsen and M. J. Harrold, "Slicing object-oriented software," in *Proc. 18th Int. Conf. Softw. Eng.*, Berlin, Germany, Mar. 1996, pp. 495–505. [Online]. Available: http://portal.acm.org/citation.cfm?id=227726.227837

[20] J. B. Gauthier, J. Desrosiers, and M. E. Lübbecke, "Decomposition theorems for linear programs," *Oper. Res. Lett.*, vol. 42, no. 8, pp. 553–557, 2014. doi: 10.1016/j.orl.2014.10.001.

[21] J. Cheng, "Slicing concurrent programs—A graph-theoretical approach," in *Proc. 1st Int. Workshop Automated Algorithmic Debugging*, 1993, pp. 223–240. [Online]. Available: http://dl.acm.org/citation.cfm?id=646902.710201

[22] N. R. Jennings, "Cooperation in industrial systems," in *Proc. ESPRIT Conf.*, Brussels, Belgium, 1991. [Online]. Available: https://eprints.soton.ac.uk/252216/1/ESPRIT-CONF91.pdf

[23] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, vol. 1. Boston, MA, USA: Addison-Wesley, 1999.

[24] L. L. Hanzo, O. Alamri, M. El-Hajjar, and N. Wu, *Near-Capacity Multi-Functional MIMO Systems: Sphere-Packing, Iterative Detection Cooperation*, vol. 4. Hoboken, NJ, USA: Wiley, 2009.

[25] Y. Zou, X. Su, and Y. Niu, "Event-triggered distributed predictive control for the cooperation of multi-agent systems," *IET Control Theory Appl.*, vol. 11, no. 1, pp. 10–16, Jun. 2016.

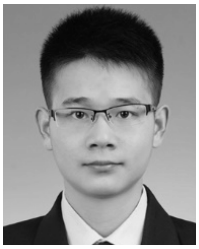
[26] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.

[27] Y.-X. He, D.-Z. Li, J.-J. Wei, L.-F. Yang, and Q. Cai, "Decomposition analysis of industrial power demand of china based on panel data model," in *Proc. 4th Int. Conf. Wireless Commun., Netw. Mobile Comput.*, Oct. 2008, pp. 1–4.

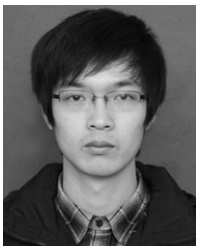
[28] F. Hsieh, "Developing cooperation mechanism for multi-agent systems with Petri nets," *Eng. Appl. Artif. Intell.*, vol. 22, nos. 4–5, pp. 616–627, 2009. doi: 10.1016/j.engappai.2009.02.006.

[29] J. Schumann, H. Jafari, T. Pressburger, E. Denney, W. Buntine, and B. Fischer, "Autobayes program synthesis system users manual," NASA Ames Res. Center, Mountain View, CA, USA, Tech. Rep. NASA/TM-2008-215366, 2008.

[30] R. Valerdi and P. Laplante, "Better requirements decomposition guidelines can improve cost estimation of systems engineering and human systems integration," Syst. Eng. Adv. Res. Initiative, Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep., 2010. [Online]. Available: http://seari.mit.edu/documents/presentations/CSER10_Liu_MIT.pdf
[31] A. Herberg and U. Lindemann, "A different view on system decomposition - subsystem-centered property evaluation in multiple supersystems," in *Proc. CSDM*, 2013, pp. 153–165. [Online]. Available: http://ceur-ws.org/Vol-1085/14-paper.pdf
[32] T. Parr, *The Definitive ANTLR 4 Reference*, 2nd ed. Raleigh, NC, USA: Pragmatic Bookshelf, 2013.



JIAWEN XIONG received the B.S. degree in software engineering from Donghua University, Shanghai, China, in 2015. He is currently pursuing the Ph.D. degree in computer science with East China Normal University, Shanghai. His research interests include model-driven development, model checking, runtime verification, program analysis, and related applications for safety-critical software systems.



JU LI received the master's degree in computer science and software engineering from East China Normal University in 2018. Before 2016, he mainly studied PowerLink protocol in the area of industrial control. His main research direction is the design and analysis of formal models in the field of industrial control.

JIANQI SHI received the B.S. degree in software engineering and the Ph.D. degree in computer science from East China Normal University, Shanghai, China, in 2007 and 2012, respectively, where he is currently with the School of Computer Science and Software Engineering, as an Associate Researcher.

From 2012 to 2014, he was a Researcher Fellow with the National University of Singapore. In 2014, he was a Research Scientist with the Temasek Laboratory under the Ministry of Defense of Singapore. His research interests include formal method, formal modeling and verification of real-time or control systems, and IEC 61508, IEC 61131 standards.

Dr. Shi's awards and honors include the Shanghai Science and Technology Committee Rising-Star Program (2018) and the ACM&CCF nomination of excellent doctor in Shanghai (2014).

YANHONG HUANG received the B.S. degree in software engineering and the Ph.D. degree in computer science from East China Normal University, Shanghai, China, in 2009 and 2014, respectively.

In 2012, she was a Research Student with Teesside University, U.K. She has been with the School of Computer Science and Software Engineering, East China Normal University, as an Assistant Researcher. Her research interests include formal method, semantics theory, analysis and verification of embedded systems, and industry software.

Dr. Huang's awards and honors include the National Scholarship (2013), the IBM China Excellent Students (2013), and the Shanghai Excellent Graduates (2009, 2014).