UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JOÃO HENRIQUE VALENTINI

INTERDEPENDENT DISTINGUISHING SENSORS AND THEIR APPLICATIONS ON DISCRETE EVENT SYSTEMS CONTROL

PATO BRANCO

2024

JOÃO HENRIQUE VALENTINI

INTERDEPENDENT DISTINGUISHING SENSORS AND THEIR APPLICATIONS ON DISCRETE EVENT SYSTEMS CONTROL

Sensores distinguidores interdependentes e suas aplicações em sistemas de controle a eventos discretos

Dissertação apresentada como requisito para obtenção do título de Mestre em Engenharia Elétrica do Programa de Pós-graduação em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Marcelo Teixeira

PATO BRANCO
2024

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Pato Branco

JOAO HENRIQUE VALENTINI

INTERDEPENDENT DISTINGUISHING SENSORS AND THEIR APPLICATIONS ON DISCRETE EVENT SYSTEMS CONTROL

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Sistemas E Processamento De Energia.

Data de aprovação: 23 de Fevereiro de 2024

Dr. Marcelo Teixeira, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Andre Bittencourt Leal, Doutorado - Fundação Universidade do Estado de Santa Catarina (Udesc)

Dr. Cesar Rafael Claure Torrico, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Marcelo Rosa, Doutorado - Universidade Federal de Santa Catarina (Ufsc)

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 23/02/2024.

Agradeço a Deus, à minha família e a todos
que me apoiaram nessa caminhada.

ACKNOWLEDGEMENTS

À todos que de alguma forma me apoiaram, em especial aos meus pais, amigos e demais familiares, pela compreensão nos momentos em que deixei de dar atenção e estar presente com eles. Estendo meus agradecimentos aos meus orientadores e demais colegas que me ajudaram a concluir mais essa etapa acadêmica em minha vida.

À UTFPR e a todos os técnicos administrativos e professores pelo apoio e ensinamentos.

Agradecimento especial ao meu colega e amigo Tiago Possato pelo apoio e parceria. À minha namorada e parceira de estudos Christine Maria Kaiser Leitner. À minha professora de inglês e amiga Gyulia Parizotto. Ao meu sócio de trabalho Messias Miranda, pela compreensão nos momentos que deixei de dar foco em nossa empresa, em razão do mestrado. Aos meus pais Reinaldo Valentini e Jane Bianchetti Valentini, pelos ensinamentos de vida e apoio financeiro que viabilizaram todas as conquistas educacionais de minha vida.

Este estudo foi financiado em parte pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código Financeiro 001, do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

RESUMO

VALENTINI, João Henrique. Sensores distinguidores interdependentes e suas aplicações no controle de Sistemas a Eventos Discretos. 2024. 37 f. Dissertação. Programa de Pós-Graduação em Engenharia Elétrica. Pato Branco. 2024.

Um *sensor distinguidor* (DS) é um artefato formal que funciona virtualmente como um sensor. Seu objetivo é fornecer detalhes sobre certos eventos em um Sistema a Eventos Discretos (DES) de acordo com seu contexto de ocorrência, o que permite a codificação de aspectos cognitivos de um DES. Às vezes, porém, certas distinções de eventos podem depender de outras distinções, fornecidas por outros DSs, em uma cadeia de dependência de contexto que torna o design do sensor em si complexo. Este artigo apresenta os fundamentos teóricos para múltiplos DSs interdependentes serem projetados por Máquinas de Estado Finito (FSM) e integrados dentro do mesmo escopo de modelagem de DES. Como essa abordagem pode resultar em modelos muito grandes, também mostramos como sintetizar supervisores usando o framework de Controle Modular Local (LMC) combinado com síntese baseada em abstração e extensões relacionadas para acomodar DSs interdependentes. Após a síntese, modelos grandes de DS geralmente sobrecarregam também a fase de implementação, uma vez que precisam ser integrados ao controlador. Portanto, este trabalho também propõe uma estratégia de implementação baseada em Internet das Coisas (IoT) que trata os DSs como artefatos de software acoplados que podem ser separados da implementação do controlador. Dessa forma, a composição unificada de controladores e DSs nunca é realmente necessária, os DSs conceituais atuam como sensores físicos, informando o controlador apenas e sempre que necessário. Uma contribuição adicional deste trabalho é o desenvolvimento e teste de uma ferramenta que converte controladores e DSs de FSMs em código implementável em C ou Python, com suporte para síntese monolítica e modular. A abordagem é ilustrada usando um controle real de despacho de uma empresa avícola.

Palavras-chave: sistemas a eventos discretos; automação discreta; sensores distinguidores; controle dependente de contexto.

ABSTRACT

VALENTINI, João Henrique. Interdependent distinguishing sensors and their applications on Discrete Event Systems Control. 2024. 37 f. Dissertação. Programa de Pós-Graduação em Engenharia Elétrica. Pato Branco. 2024.

A *distinguishing sensor* (DS) is a formal artefact which works virtually like a sensor. Its purpose is to provide details about certain events in a Discrete Event System (DES) according to their context of occurrence, which allows the coding of cognitive aspects of a DES. Sometimes, however, certain event distinctions may depend on other distinctions, provided by other DSs, in a context dependency chain that makes the design of the sensor itself complex. This paper presents the theoretical foundation for multiple interdependent DSs to be designed by *Finite State Machines* (FSM) and integrated within the same DES modelling scope. As this approach may result in very large models, we also show how to synthesise supervisors using the *Local Modular Control* (LMC) framework combined with abstraction-based synthesis and related extensions to accommodate interdependent DSs. After synthesis, large DS models usually overload also the implementation phase, since they have to be integrated with the controller. Therefore, this work also proposes a *Internet of Things* (IoT)-based implementation strategy that treats DSs as loose-coupled software artefacts that can be separated from the controller implementation. In this way, the unfolded composition of controllers and DSs is never actually needed, and the conceptual DSs act like physical sensors, informing the controller only and whenever required. A further contribution of this work is developing and testing a tool that converts FSMs controllers and DSs into implementable code C or Python, with support for both monolithic and modular synthesis. The approach is illustrated using a real dispatch control of a poultry company.

Keywords: discrete event systems; discrete automation; distinguishing sensors; context-dependent control.

LIST OF FIGURES

CONTENTS

1 INTRODUCTION

Modern directions of industry subsume the intensive integration of factory components with information processing. The appropriate link between devices and information approximates the factory floor from business intelligence and enables emerging production models.

Measurement and instrumentation technologies are essential components for the integration between devices and computers. They allow sensing, (edge-)processing, and storing variables from the factory floor to the cloud without human intervention. However, some variables may not be directly sensed by electronic devices, either due to the lack of precision, the uncertainty of the environments, the unmeasurable nature of variables, the inherent cost, etc. In these cases, variable values may have to be derived after computational processing. Examples include friction and heat transfer indexes in industrial processes, composite navigation for autonomous robots (AFONSO; FERREIRA, 2022), context-aware memory for manufacturing systems (SILVA; RIBEIRO; TEIXEIRA, 2017), and others.

Some computational techniques can be applied to calculate variable values not directly read by sensors. Sensory fusion, for example, uses multiple values measured from different sensors and derives a new composite value (LUCA; ORIOLO; GIORDANO, 2007). Inversely to the idea of fusion, some problems may require expanding the information details on the occurrence of certain phenomena. A sensor on a production line, for example, can detect the presence of components, but it cannot recognize, for instance, how many times a given component has been detected. This is more naturally recorded by models that simultaneously handle multiple variable contexts.

A modelling approach that fits this purpose is proposed in (CURY *et al.*, 2015). It uses a Discrete Event System (DES) model called a *distinguishing sensor* (DS), which behaves virtually like a sensor that tracks sequences of DES events and determines, for each sequence, to which operation context it belongs. In the previous example, a distinguisher could recognize, in addition to the presence of a component, its input source or how many times it has been detected so far.

Despite their usefulness, the core principle of DSs assumes that each event in a DES emerges from a single context of occurrence in the plant, and no dependency from other events is needed to distinguish it, restricting the use of DSs to some applications. In this work, DSs are looked at from the perspective that they could be restructured to handle in parallel multiple interdependent contexts. In the previous example, it might be the case to detect: (i) the presence of a component (physical sensor); (ii) how many times each component was detected (DS 1 detailing the physical sensor); and (iii) the buffering position of components detected multiple times (DS 2 using distinctions of DS 1).

The contributions of this work can therefore be summarized as follows. We present the theoretical foundation for multiple interdependent DSs to be designed by *Finite State Machines* (FSM) and integrated within the same DES modelling scope (VALENTINI *et al.*, 2023). As

this approach usually results in very large models, instead of adopting a monolithic approach to the synthesis process, we use a local modular method that combines the *Local Modular Control* (LMC) framework (QUEIROZ; CURY, 2000) with the abstraction-based synthesis process (CURY *et al.*, 2015). Within this local procedure, the necessary plant models for each synthesis step are selected based on sharing events with the specification models, whereas the selection of distinguisher models relies on a controllability criterion (ROSA; TEIXEIRA; MALIK, 2018). As a result, supervisors are individually computed through a local synthesis process over a reduced set of plants and DSs.

As the implementation phase is also impacted by large DS models that have to be integrated with the controller, we also propose an *Internet of Things* (IoT)-based implementation strategy that treats DSs as loose-coupled software artefacts that can be separated from the controller implementation. The link between the modelling interface and implementation is paved by the development and testing of a tool that converts FSMs controllers and DSs into implementable code C or Python, with support for both monolithic and modular synthesis (POSSATO, 2023).

For illustration, the approach is applied to the dispatch control of a broiler meat industry. In this process, identical packages, with different content, are transported on mobile conveyors. The purpose is to automate the separation according to their type and origin. Two DSs are added to the exit point of the conveyors for this purpose, one to memorize where each package emerges from and transmit this information to the second sensor, which shifts the origin identification according to the positions occupied by each package along the line. As a result, it is possible to recognize the profile information of each package that enters the dispatch line at any point of the workflow and thus define its destination properly. Such a solution would be unfeasible to be programmed manually, considering the huge effect of combining events and states in memory and time processing.

In the following, Section 2 presents the theoretical background; Section 3 introduces the interdependent DSs; the application is detailed on Section 4; the code generator is described in the Section 5; the case study is revisited in Section 6; and some conclusions and perspectives are discussed in Section 7.

2 FOUNDATIONS OF DISCRETE EVENT SYSTEMS

In *Discrete Event Systems* (DESs) (CASSANDRAS; LAFORTUNE, 2021), state transitions do not depend continuously on time but on asynchronous events that occur irregularly, in a step-by-step manner. Physical, biological, and social phenomena are usual examples of DESs.

Modeling a phenomenon as a DES helps to understand and find ways to improve it. *Strings* are usual structures to describe the sequence of steps of a DES. Each step is labelled by an *event* σ taken from an *alphabet* Σ, where Σ^* denotes the set of all finite strings formed with events in Σ, including the *empty string* ε. Any subset $L \subseteq \Sigma^*$ is called a *language*. Two strings $s,t \in \Sigma^*$ can be *concatenated* as st and the *prefix-closure* $\overline{L}$ of a language L is the set of all prefixes of L, that is, $\overline{L} = \{s \in \Sigma^* \mid st \in L \text{ for some } t \in \Sigma^*\}$.

In practice, it is convenient to generate languages from diagrams more tuned with design tasks, such as the *Finite-State Machines* (FSMs). A FSM is a 5-tuple $G = (\Sigma,X,\rightarrow,x^\circ,X^m)$, where Σ is the set of events, X is the set of states, $\rightarrow: X \times \Sigma \rightarrow X$ is the partial state transition function, x° is the initial state, and X^m is the set of marked states, representing tasks completed by the DES.

For two states $x_1,x_2 \in X$, $x_1 \xrightarrow{\sigma} x_2$ denotes a transition from the state x_1 to x_2 with the event $\sigma \in \Sigma$. The same notation is generalized for strings $s \in \Sigma^*$, i.e., $x_1 \xrightarrow{s} x_2$ means that x_2 is reached from x_1 with s, while $G \xrightarrow{s} x$ means that a string s is possible in G from its initial state to reach state x.

The language generated by G is denoted by $L(G) \subseteq \Sigma^*$ and defined as $L(G) = \{s \in \Sigma^* \mid G \xrightarrow{s} x \in X\}$. It represents the set of all the strings of events that can occur in G. Differently, $L^m(G) \subseteq L(G)$ is defined as $L^m(G) = \{s \in \Sigma^* \mid G \xrightarrow{s} x \in X^m\}$ and denotes the language marked by G, i.e. those strings that lead the DES to complete tasks.

Given two FSM components $G^1 = (\Sigma_1,X_1,\rightarrow_1,x_1^\circ,X_1^m)$ and $G^2 = (\Sigma_2,X_2,\rightarrow_2,x_2^\circ,X_2^m)$, their *synchronous composition* (CASSANDRAS; LAFORTUNE, 2021), denoted $G^1 \| G^2$, is an operation that: merges events that are enabled by both G^1 and G^2; interleaves (in any order) events that are enabled by only one FSM and unknown by the other; and disables events that are enabled by one FSM and known, but not enabled, by the other. This is the main mechanism for imposing restrictions on a DES by modelling. A state is considered to be marked after composition if the corresponding combination of states is also marked in the input FSMs.

2.1 DES Control

A DES often includes $n \geq 1$ components that are expected to work together. Each component can be locally modelled as an FSM G^j, and the DES open-loop behavior can be given by the composition $G = G^1 \| \ldots \| G^n$, which is called the *plant*. In conjunction, FSMs G^j, for $j = 1, \cdots, n$, are expected to interact properly with each other, and it is the role of a

controller to define how and when they do. In *Supervisory Control Theory* (SCT) (RAMADGE, 1989), a DES controller (called *supervisor*) can be computed automatically from a set of FSMs, as long as control specifications are provided.

Specifications are rules that impose prohibitive actions to G, e.g., the priority of one component over another. Analogously to G, a specification E can also result from the composition of modules E^i, i.e., $E = E^1 \| \ldots \| E^m$, for $m \geq 1$. It can be joined to G to form the *desired behaviour* under control, denoted $K = L(G \parallel E)$. In the following, we will refer to K either as a language or the FSM $G \parallel E$ that generates K, whenever its role is clear for the context.

In order to form a proper *supervisor* as in SCT, the set of events of G must be split into $\Sigma = \Sigma^c \dot{\cup} \Sigma^u$, where the ones in Σ^c can be inhibited by control, while the supervisor cannot directly prevent those in Σ^u. A necessary and sufficient condition for the existence of a supervisor is the *controllability*: for a prefix-closed plant behaviour $L(G) \subseteq \Sigma^*$, a specification $K \subseteq L(G)$ is controllable with respect to $L(G)$ if $\overline{K}\Sigma^u \cap L(G) \subseteq \overline{K}$.

That is, K is controllable if it enables the uncontrollable events whenever they are eligible in the plant. When K is not controllable, it can be reduced iteratively to its largest (or supremal) controllable sublanguage. This computation is called *control synthesis*, and $\sup\mathcal{C}(K,G)$ is the usual notation for the resulting great supervisor whose FSM is denoted here by S.

In practice, S represents the most permissive set of actions to be imposed by control on G while complying with controllability and the specification. If $S\|G$ is also *nonblocking* (CASSANDRAS; LAFORTUNE, 2021), that is, $L(S) = \overline{L_m(S)}$, it is said to be *optimal*. A nonblocking supervisor S ensures that, regardless of the actions (events) executed in the plant under its control, there always exists a sequence of next actions leading to a situation of task completion (marked states).

Implementing S adds advantages compared to the empirical programming of controllers, as it automates (complex) manual tasks and leads to correct-by-construction codes. However, the SCT faces strong limitations in synthesizing controllers using monolithic algorithms, as their complexity grows exponentially with the number of components of a DES. This prevents the SCT from being applied over real-scale DESs. Modular alternatives are discussed in the following.

2.2 Local Modular synthesis

When the multiple plants and specifications components are synchronized together and this results in the synthesis of a single controller, the procedure is said to be *monolithic*.

In monolithic synthesis, the *state-space explosion* problem may become a barrier for SCT to be adopted in the industry. Modular approaches have tried to simplify the monolithic procedure by exploiting only subsets of components in an attempt to construct the optimal supervisor, reducing computational complexity in supervisor synthesis.

Among the options of modular framework (MOHAJERANI; MALIK; FABIAN, 2017; MALIK; TEIXEIRA, 2021), the LMC (QUEIROZ; CURY, 2000; ÅKESSON; FLORDAL; FABIAN, 2002; MALIK; TEIXEIRA, 2016) appears a straightforward alternative. Among the advantages of a modular system compared to a monolithic one, we can mention that it allows different parts of the controller to be updated, replaced, or modified independently of each other. Additionally, it provides ease in identifying and isolating specific problems. It consists of obtaining a local controller for each specification, which improves (WONHAM; RAMADGE, 1988) by calculating it using only plants affected by the events of that particular specification.

Formally, let E^i, for $i \in I = \{1, \cdots, m\}$, be specifications modeled with events in $\Sigma_{E^i} \subseteq \Sigma$; and G^j, for $j \in J = \{1, \cdots, n\}$, be plants designed with events in Σ_{G^j}. For $i = 1, \cdots, m$, the local plants G_i^{loc}, associated with each specification E^i, are

$$\mathrm{G}_i^{loc} = \|_{j \in J_i} G^j,$$

such that

$$J_i = \{j \in J \mid \Sigma_{G^j} \cap \Sigma_{E^i} \neq \emptyset\}.$$

That is, each G_i^{loc} includes only plants that had at least one event used to design E^i. The remaining steps of synthesis follow as usual: the composition $\mathrm{G}_i^{loc} \parallel E^i$ leads to K_i^{loc} and $\sup\mathcal{C}(\mathrm{G}_i^{loc}, \mathrm{K}_i^{loc})$ is a supervisor modelled by S_i^{loc}.

As the synthesis of each S_i^{loc} can be seen as an instance of the monolithic procedure, S_i^{loc} is certainly nonblocking. However, in conjunction, supervisors S_i^{loc} may conflict with each other and result in a blocking behaviour of the plant under control. In (QUEIROZ; CURY, 2005) and older results there are conditions to test and guarantee the non-conflict of local controllers, which are hidden here for the sake of brevity. In summary, when the non-conflicting condition is achieved, the joint action of local controllers over the plant is optimal, that is, if the set $\{\mathrm{S}_i^{loc}, i = 1, \cdots, m\}$ is non-conflicting, then $S = \|_{i=1}^{m} \mathrm{S}_i^{loc}$.

Despite clear improvements added to the monolithic SCT by the LMC, they both are limited when the coordination of a DES depends on handling multiple contexts of a DES in parallel. The concept of DSs, presented next, complements well the SCT (CURY *et al.*, 2015) and the LMC (TEIXEIRA; CURY; QUEIROZ, 2018) towards this direction.

2.3 Distinguishing Sensors

A *distinguishing sensor* (DS) (CURY *et al.*, 2015) can be seen as a complementary part of a FSM that models a DES plant. Its purpose is to provide more details about some events in the plant model so that control actions can be better engineered.

Formally, when a plant G, defined over a set of events Σ, is associated with a DS, each event $\sigma \in \Sigma$ is associated with a nonempty set of new events Δ^σ, known as *refinements*, each

one corresponding to a different instance of the occurrence of σ in the system. Refined events in Δ^σ inherit the controllability of σ, that is, events in Δ^σ are (un)controllable if and only if σ is (un)controllable. Thus, $\Delta^c = \bigcup_{\sigma\in\Sigma^c} \Delta^\sigma$, $\Delta^u = \bigcup_{\sigma\in\Sigma^u} \Delta^\sigma$ and $\Delta = \bigcup_{\sigma\in\Sigma} \Delta^\sigma$ represents the set of refined events. Analogously to Σ^*, Δ^* denotes the set of all possible *refined strings*, and their relationship relies on mappings. Map $\Pi : \Delta^* \to \Sigma^*$ transforms refined strings into the corresponding input string, while the inverse map Π^{-1}: $\Sigma^* \to 2^{\Delta^*}$ transforms each input string into refined strings. The recursive definition of Π and Π^{-1}, details, and examples can be found in (CURY *et al.*, 2015).

Both Π and Π^{-1} can be straightforwardly extended to languages or FSMs. For any language $L_d \subseteq \Delta^*$,

$$\Pi(L_d) = \{s \in \Sigma^* \,|\, \exists t \in L_d,\, \Pi(t) = s\}.$$

Similarly, for a language $L \subseteq \Sigma^*$,

$$\Pi^{-1}(L) = \{t \in \Delta^* \,|\, \Pi(t) \in L\}.$$

The effect of a DS can then be formally introduced. For $L \subseteq \Sigma^*$, a DS is a map $D : \Sigma^* \to 2^{\Delta^*}$ such that

$$D(L) = \Pi^{-1}(L) \cap L_d\,. \tag{1}$$

That is, D transforms each string of the input model L into refined strings ($\Pi^{-1}(L)$) and filters those strings using an auxiliary, prefix-closed, language $L_d \subseteq \Delta^*$. L_d chooses, among all possible refined strings, those which should really be eligible. For a language $L_d = D(\Sigma^*)$, $\Pi(L_d) = \Sigma^*$ and $\Pi(D(L)) = L$ are properties that follow from (CURY *et al.*, 2015).

Remark that Π^{-1} is a particular case of D when L_d does not impose any filtering, i.e., $L_d = \Delta^*$. Inversely, another particular case is when L_d filters as a one-to-one relationship, i.e., every string $s \in \Sigma^*$ is mapped into exactly one $r \in \Delta^*$, and then $|D(s)| = 1$. In this case, D is said to be *predictable*, and it is the only case assumed in this paper.

As for FSMs, the effect of $\Pi^{-1}(G)$ means simply replacing the label of each transition in G with the corresponding sets of refined labels. That is, $G \xrightarrow{s} x \xrightarrow{\sigma} x'$ implies $\Pi^{-1}(G) \xrightarrow{t} x \xrightarrow{\delta} x'$ for all $t \in \Pi^{-1}(s)$ and $\delta \in \Delta^\sigma$. Thus, $D(G) = G_d = \Pi^{-1}(G) \| H_d$ means additionally filtering those sets of refined labels by one, and only one, label elected to survive in G_d. This choice is an assignment of the model H_d, which models L_d. That is, $G \xrightarrow{s} x \xrightarrow{\sigma}$ implies $G_d \xrightarrow{r=D(s)} x_d \xrightarrow{\delta\in\Delta^\sigma}$.

For a DES modelled by G, let E_d be the version of the specifications designed with events in Δ; let H_d be the distinguisher model for events Δ; and let $G_d = \Pi^{-1}(G) \parallel H_d$ be the plant version in Δ. The desired behaviour is now modelled as $K_d = L(G_d \parallel E_d)$. As H_d is predictable, the SCT can be equivalently processed either using G_d and K_d, or G and $K = \Pi(G_d \parallel E_d)$. According to (CURY *et al.*, 2015) that the respective results, $\sup\mathcal{C}(K_d, G_d)$ and $\sup\mathcal{C}(K, G)$, are equivalent.

2.4 Approximations in control with DSs

The predictable DS model generally emerges from the composition of smaller sub-DS models. However, this composition can lead to a very large model, in terms of state-space, that can be hard to process, especially through monolithic synthesis. In (CURY *et al.*, 2015), the authors suggest the possibility of removing part of the DS model from the synthesis procedure, as a way to alleviate processing.

Formally, let H_d be the model of a predictable DS for the plant G, such that $G_d = \Pi^{-1}(G) \parallel H_d$ follows as in Section 2.3. Now, let another distinguisher model H_a be such that $L(H_d) \subseteq L(H_a) \subseteq \Delta^*$. Then, any plant $G_a = \Pi^{-1}(G) \parallel H_a$ is an approximation for G_d and synthesis is processed over $K_a = G_a \parallel E_d$ to obtain a supervisor S_a that models $\sup\mathcal{C}(K_a, G_a)$.

By choosing G_a as a plant, instead of G_d, one may expect that it has fewer states than G_d. For example, the approximation $G_a = \Pi^{-1}(G) \parallel H_a$, for $L(H_a) = \Delta^*$, has the same number of states as G. However, it can be shown (CURY *et al.*, 2015) that $\sup\mathcal{C}(K_a,G_a) \cap L(H_d) \subseteq \sup\mathcal{C}(K_d,G_d)$ and, therefore, the result may be suboptimal. In fact, by not relying on the information provided by H_d, the synthesis with G_a may have to restrict actions in the plant that eventually lead it to reach unsafe states, resulting in a supervisor S_a that tends to be more restrictive than S_d.

2.5 Suitable approximations in control with DSs

Conditions to make sure that $\sup\mathcal{C}(K_a,G_a) \cap L(H_d) = \sup\mathcal{C}(K_d,G_d)$ are provided in (ROSA; TEIXEIRA; MALIK, 2018) through the notion of *target events*. An event is said to be target if it is uncontrollable and if, after a given sequence in the plant, the controller fails to determine whether a progression towards that event leads the system to a certainly safe or a certainly unsafe state. Under uncertainty, the controller is forced to disable the entire progression, violating the maximal permissiveness to keep safety.

An option to eliminate target events is by adding to G_a the DS models that distinguish them. As DS model is usually given as a set of modular FSMs, the DS modules that distinguish target events can be simply composed with G_a. Once no target events exist, G_a is an approximation that is sufficient in the least restrictive controllable synthesis. In Section 4.7 we discuss how this notion of suitable approximation applies to the LMC.

3 INTERDEPENDENT DISTINGUISHING SENSORS

In some cases, it may happen that more than one DS is needed at different steps of a DES progression. When these steps are independent or manipulate independent contexts in the plant, the DS principles can be merely replicated n times.

However, when two or more steps need to be detailed in the same stage of the progression of a DES, the DS can to be designed by two or more sets of models that relate each level of interdependent elaboration. This ensures that the state machines of the DSs generate the expected context for the designer.

In these cases, the whole control problem relies on a cognitive structure that recognizes the interdependence between DSs, which has not been explored in the literature so far.

In order to systematize this notion, we now present a theoretical foundation to associate contexts step-by-step through interdependent chains. Formally, for an input event set Σ and its firsthand refined version Δ_1, such that

$$\forall \delta \in \Sigma, \exists \Delta_1^{\delta} \subseteq \Delta_1,$$

let Δ_2 be a refinement set for Δ_1, such that

$$\forall \delta_1 \in \Delta_1, \exists \Delta_2^{\delta_1} \subseteq \Delta_2\,;$$

and Δ_k be the refinement of Δ_{k-1}, such that

$$\forall \delta_{k-1} \in \Delta_{k-1}, \exists \Delta_k^{\delta_{k-1}} \subseteq \Delta_k\,.$$

In summary, departing from an original set of events, which reflects the set of observable signals in a DES, interdependent levels of distinctions can be applied systematically over this set to recognize a new context at each step.

The chained relationship between the sets Σ,$\Delta_1, \cdots, \Delta_k$ can be straightforwardly extended as follows:

$$\Pi_1 : \Delta_1^* \rightarrow \Sigma^*\,;$$

$$\Pi_2 : \Delta_2^* \rightarrow \Delta_1^*\,;\ ...$$

$$\Pi_k : \Delta_k^* \rightarrow \Delta_{k-1}^*\,.$$

Inversely:

$$\Pi_1^{-1} : \Sigma^* \rightarrow 2^{\Delta_1^*}\,;$$

$$\Pi_2^{-1} : \Delta_1^* \rightarrow 2^{\Delta_2^*}\,;\ ...$$

$$\Pi_k^{-1} : \Delta_{k-1}^* \to 2^{\Delta_k^*}.$$

Fig. 1 provides a visual overview of the mappings.

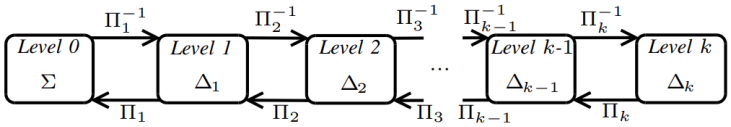

Figure 1 – Context mapping for multiple interdependent DSs**.**

We can now associate a DS to handle each level of information expanded by those chained mappings. For a language $L \subseteq \Sigma^*$, a sensor D_k for level k is recursively defined by

$$D_k(L) = \begin{cases} \Pi_1^{-1}(L) \cap L_{d_1} & \text{if } k = 1, \\ \Pi_k^{-1}(D_{k-1}(L)) \cap L_{d_k} & \text{if } k > 1. \end{cases}$$

At the first level of information, the sensor D_1 follows from Eq. (1). As for the second level, the effect of a sensor $D_2 : \Delta_1^* \to 2^{\Delta_2^*}$ receives as input an already distinguished language $D_1(L) \subseteq \Delta_1^*$ and returns its next level of refined information. The same idea applies to the other levels.

Analogously, the effect of D_k idem over a plant G is given by

$$D_k(G) = \begin{cases} \Pi_1^{-1}(G) \| H_1 & \text{if } k = 1, \\ \Pi_k^{-1}(D_{k-1}(G)) \| H_k & \text{if } k > 1. \end{cases}$$

In this way, the plant G_d can be constructed for G through the incremental application of k-steps distinctions, so that $L(G_d) = D_k(L(G))$ and $L_m(G_d) = D_k(L_m(G))$.

Anologously, a distinguisher $D_k : \Delta_{k-1}^* \to 2^{\Delta_k^*}$ is predictable for level k if every string in Δ_{k-1}^* is mapped into exactly one string in Δ_k^*. Formally, $|D_k(s_{k-1})| = 1$, for all $s_{k-1} \in \Delta_{k-1}^*$. It can be shown that if strings are mapped predictably between any two consecutive levels in $\{1, \ldots, k\}$, the entire mapping chain from level 1 to level k remains predictable.

Proposition 1. *Let $D_1, \ldots, D_k$ be interdependent distinguishers for levels $1, \ldots, k$, respectively. If all D_i, $i \in \{1, \ldots, k\}$ are predictable for level i, then $|D_k(s)| = 1$ for all $s \in \Sigma^*$.*

Proof. The proof of this proposition is an induction on the level of distinction starting with $i = 1$.

- Basis: let $i = 1$, then $D_1 : \Sigma^* \to 2^{\Delta_1^*}$. Thus, by assumption that D_1 is predictable for level 1 it follows that $|D_1(s)| = 1$ for all $s \in \Sigma^*$.

- Induction: for the inductive hypothesis let us assume that $|D_i(s) = 1|$ for all $s \in \Sigma^*$. Let $s_i \in D_i(s)$ and $D_{i+1} : \Delta_i^* \rightarrow 2^{\Delta_{i+1}^*}$. Note that

$$\begin{aligned} D_{i+1}(s) &= \Pi_{i+1}^{-1}(D_i(s)) \cap L_{d_{i+1}} \\ &= \Pi_{i+1}^{-1}(s_i) \cap L_{d_{i+1}} \end{aligned} \tag{2}$$

  Therefore, from the assumption that D_{i+1} is predictable combined with Equation (2) it follows that

$$|D_{i+1}(s)| = |\Pi_{i+1}^{-1}(s_i) \cap L_{d_{i+1}}| = 1$$

□

It can be shown that a control solution obtained from a set of interdependent predictable distinguishers $D_1, \ldots, D_k$ is equivalent to one that would be obtained without using refined events and distinguishers.

Corollary 2. *For a* DES *modeled by* G *and a set of interdependent predictable distinguishers* $D_1, \ldots, D_k$*, let* E_d *be the version of the specification designed in* Δ_k *and* $G_d = D_k(G)$ *be the plant version in* Δ_k*, such that* $K_d = L(G_d \| E_d)$*. Then*

$$\begin{gathered} \Pi_k(\sup\mathcal{C}(K_d, G_d)) = \sup\mathcal{C}(K, G) \\ \textit{and} \\ \sup\mathcal{C}(K_d, G_d) = D_k(\sup\mathcal{C}(K, G)) \end{gathered} \tag{3}$$

where $K = \Pi_k(K_d)$.

Proof. The proof of this corollary straightforwardly follows from Proposition 1 in conjunction with Theorem 2 introduced in (CURY *et al.*, 2015). □

Example 1. *Consider the partial* DES *in Fig. 2.*

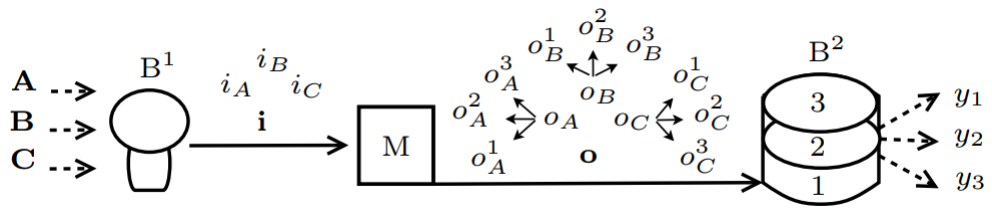

Figure 2 – Example of DES **with interdependent contexts.**

The system receives external components in a buffer B^1*, coming from 3 different sources, Buffer* A*, Buffer* B*, and Buffer* C*, each source carrying a different context of manufacturing.*

The meaning of each context is not important here; it could be related to the origin through the workflow, the type of components, the number of times the same component reached B^1*, or any other descriptor of interest. Machine* M *removes components from* B^1 *(event* i*) and is expected to finish working with event* o*. However, it is crucial for* M *to recognize and process each context accordingly, allowing customized actuation schemes to be applied.*

To map those contexts, let D_1 *be the sensor for level 1, with* $\Pi_1^{-1}(\tau) = \{\tau_A, \tau_B, \tau_C\}$ *for* $\tau \in \{i,o\}$.

From M*, components are transferred to the buffer* B^2 *of capacity 3, where they are associated with 3 more contexts intended to memorize the combinations of type and positions in* B^2*. Therefore, let us also define* D_2 *as the sensor for level 2, applied to level 1, in such a way that* $\Pi_2^{-1}(\tau_j) = \{\tau_j^1, \tau_j^2, \tau_j^3\}$ *for each* $\tau_j \in \Pi_1^{-1}(\tau)$ *and with 1, 2 and 3 mapping each position of* B^2.

From B^2*, components again take different paths, represented in Fig. 2 by the events* y_1*,* y_2 *and* y_3*. They could be associated, for example, with recycling outputs, exit points, customization paths, etc. When deciding which output to take depends on the combination of contexts carried by each component, one has to memorize and update those contexts concurrently throughout the system to model such a decision as a specification.*

It can be shown (ROSA et al., 2017; CURY et al., 2015; TEIXEIRA; CURY; QUEIROZ, 2018) that this type of model requires enormous manual effort to be constructed. DSs *help to reduce such a manual effort, and they have been applied to buffering control in (ROSA et al., 2017), to rework cycles control in (CURY et al., 2015; TEIXEIRA; CURY; QUEIROZ, 2018), to customized control policies in (SILVA; RIBEIRO; TEIXEIRA, 2017), among others. However, problems considered in the literature do not consider more than one* DS *interdependently, which is expected to be more challenging. The next section presents an application that exposes this complexity and is followed by a modelling proposal inspired by the theory proposed in Section 3 and exemplified here.*

4 A DISPATCH CONTROL APPLICATION

In industries that process broiler meat, a pertinent question is how to keep track of which type of meat was included in which package to be shipped. After slaughtering, broilers are divided into pieces that receive different commercial values and attention along the processing line. Each type of meat is packaged and sent for shipping, a step which usually centralizes the production from multiple sources and routes.

The task of classifying packages for shipping can be easily automated when they differ in shape, colour, or weight. However, when packages are identical, one possibility to identify them is by attaching electronic tags, under the unreasonable effort and cost of tagging each package. Alternatively, the controller could be engineered to memorize the source of each package, carry this information throughout the process, and use it for dispatch decisions. The approach proposed in this paper is now applied for this purpose, and we use the dispatch flow in Fig. 3 for illustration.

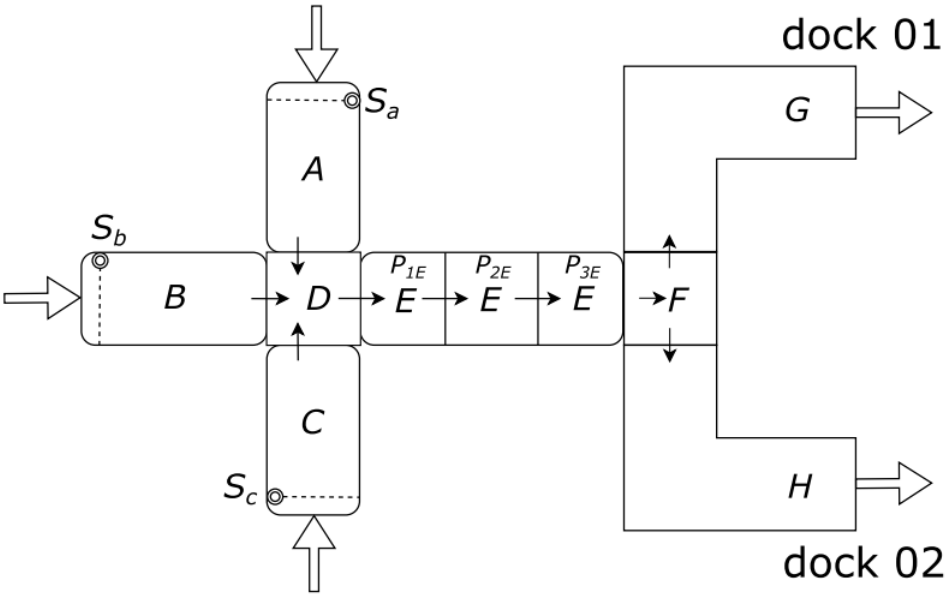

Figure 3 – Workflow of the evaluated process.

Packages are received from 3 conveyors, A, B, or C, and their arrival is signalized by sensors S_a, S_b and S_c, respectively. They are identical in shape, but their contents may differ according to where they were produced. Thus, the type of products can be classified according to the conveyor from where they enter the system, leading us to identify types A, B, and C of products. Conveyors A, B, and C move packages from the respective source to conveyor D, and from there D moves to E. But, differently from D, E has three positions, P_{1E}, P_{2E}, and P_{3E}, equally spaced, which can carry up to three packages at the same time, evolving one position at a time. Finally, conveyor F receives packages from the last position of E and releases them from the system through dock 01 (conveyor G) or dock 02 (conveyor H).

4.1 Plant modelling

Fig. 4 shows the FSMs that model each plant component.

Figure 4 – Proposed plant models.

Model G^{S_I}, for $I = A, B, C$, represent the behaviour of the sensor S_I, while G^J, for $J = A, B, C, D, E, F, G, H$, model the conveyor J. Event I indicates that sensor S_I has been triggered; events s_J model the *starting* of conveyor J, while events f_J *finishing* the movement of the conveyor J. It has been assumed that $\Sigma^c = \{s_J\}$, while $\Sigma^u = \{I, f_J\}$. The composition $G = G^{S_I} \parallel G^J$ is an FSM with 256 states and 2048 transitions.

4.2 Standard specification modelling

To enforce the expected workflow to the plant, the following specifications are considered:

$E^{1,2,3}$**:** prevent conveyor I, for $I = A, B, C$, to start before a packaging is identified by the respective S_I.

$E^{4,5}$**:** It only allows turning on conveyor D after finishing the activation of conveyor A, B, or C.

$E^{6,7,8}$**:** avoid underflow and overflow at conveyors E (E^6), F (E^7), G and H (E^8).

E^9**:** allow conveyor E to carry up to 3 packages at the same time free of overflow and underflow.

E^{10}**:** separate the packages type C to be sent to conveyor G, while the others go to conveyor H.

Fig. 5 shows the FSM model for the specifications.

Specifications E^1, E^2 and E^3 disable the starting of conveyors A, B, and C, unless the respective sensor S_A, S_B, or S_C detects the arrival of a packages. Specifications E^4, E^5. Specifications E^6, E^7 and E^8 control, respectively, the start of the conveyor D, E, F, and G or H, only after the conclusion of the previous conveyor in the flow; they also avoid a conveyor to start twice before the next conveyor in the flow removes the packeges.

While these specifications can be easily modelled using only events in Σ, E^9 and E^{10} impose a much more challenging design task. When a packages is moved from conveyor D to E, the events s_E and f_E mean simply moving the packages from position 1 to position 2 of E. As

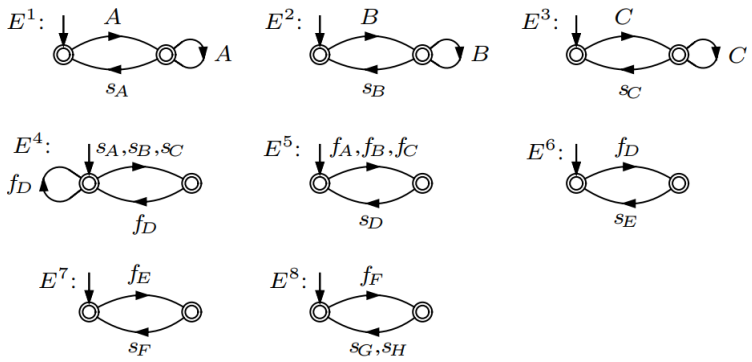

Figure 5 – Specifications for the coordination of the conveyors.

E is a 3-position conveyor, the packages is only ready to be collected by F after the event pair s_E followed by f_E still occurs twice more, finally reaching the third position. This means that specifications E^6 (coordination D-E) and E^7 (coordination E-F) work well for a 1-position conveyor but not for two or more. Therefore, E^6 and E^7 are not correctly modeled and should not be presented as models for their corresponding specifications. Model G^E could also not be extended to map positions because the physical device has only those two drive commands, s_E and f_E, independent of how many positions it has to be considered at the logical level.

In this sense, specifications E^6 and E^7 have to be reconstructed to satisfy the requirement E^9, considering more positions on E. As for E^{10}, it requires memorizing each combination of type and position throughout the conveyors to finally separate each packages in the correct dispatch dock, which is also associated with a complex design task.

4.3 Remodeling the plant with more details

An alternative to simplify this modelling task is to provide more details about specific events in Σ.

Consider that Δ_1 is a new alphabet that details the type of packages handled by the system. Then, $\Delta_1^{s_D} = \{s_{DA}, s_{DB}, s_{DC}\}$, $\Delta_1^{f_D} = \{f_{DA}, f_{DB}, f_{DC}\}$, $\Delta_1^{s_E} = \{s_{EA}, s_{EB}, s_{EC}\}$, and $\Delta_1^{f_E} = \{f_{EA}, f_{EB}, f_{EC}\}$, indicating the start and finishing of the conveyors D and E while carrying the types A, B, or C of components.

Moreover, assume that Δ_2 is a second level of detail, this time applied over events in Δ_1 to enrich them further with details about the positions occupied by each type of package in E. Then, $\Delta_2^{s_{EA}} = \{s_{EA1}, s_{EA2}, s_{EA3}\}$, $\Delta_2^{s_{EB}} = \{s_{EB1}, s_{EB2}, s_{EB3}\}$, $\Delta_2^{s_{EC}} = \{s_{EC1}, s_{EC2}, s_{EC3}\}$. Similarly, $\Delta_2^{f_{EA}} = \{f_{EA1}, f_{EA2}, f_{EA3}\}$, $\Delta_2^{f_{EB}} = \{f_{EB1}, f_{EB2}, f_{EB3}\}$, $\Delta_2^{f_{EC}} = \{f_{EC1}, f_{EC2}, f_{EC3}\}$. In this case, 1, 2, and 3 identify each position of E.

For any other event σ not influenced by these two maps, it is assumed that $\Delta_1^{\sigma} = \Delta_2^{\sigma} = \{\sigma\}$. We can now reintroduce the plant components whose events have been changed by these maps. This leads to an approximation $G_a = \Pi_2^{-1}(G)$ of the original plant G, where G_a is a

version of G defined over Δ_2. Fig. 6 shows the FSM G_a, which has 256 states, the same as G, as expected, but it has more transitions, 5376, resulting from the details added to the plant.

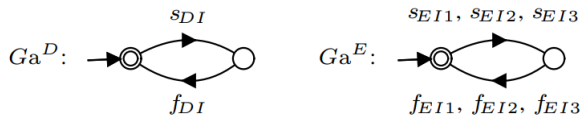

Figure 6 – Modification of the plant model to consider the maps Δ_1 and Δ_2, for $I = A, B, C$.

4.4 Specifications under the new alphabet

With new information in the plant model, the specifications can be redesigned as in Fig. 7.

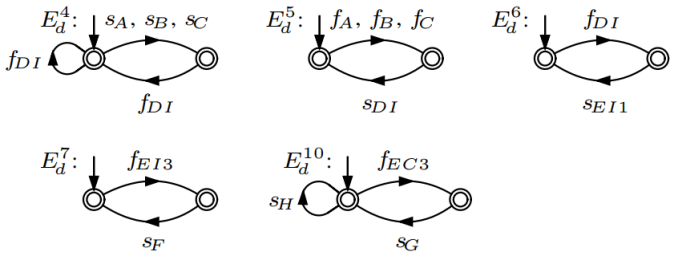

Figure 7 – Specifications redesigned according with the maps Δ_1 and Δ_2. Assume $I = A, B, C$.

Specifications $E_d^1 = E^1$, $E_d^2 = E^2$, $E_d^3 = E^3$ and $E_d^8 = E^8$ are not presented, as they remain unchanged from Fig. 5. E_d^4 and E_d^5 simply projects E^4 and E^5 considering the new alphabet, that is, $E_d^4 = \Pi_2^{-1}(E^4)$ and $E_d^5 = \Pi_2^{-1}(E^5)$. E_d^6 can now prevent the underflow and overflow from D to E by manipulating only the events of the first position of E (s_{EI1}), releasing the other conveyor's positions from this checking. Similarly, E_d^7 avoids the underflow and overflow from E to F based only on the events of the last position of E (f_{EI3}). In conjunction, E_d^6 and E_d^7 satisfy the specification E^9. Finally, as now the system knows the context of each package that reaches the last position of conveyor E, the separation of type C can be easily ensured by E_d^{10}.

The new version of the specification, $E_d = E_d^1 \parallel E_d^2 \parallel E_d^3 \parallel E_d^4 \parallel E_d^5 \parallel E_d^6 \parallel E_d^7 \parallel E_d^8 \parallel E_d^{10}$, has 512 states and 6400 transitions.

4.5 Modelling the distinguishing sensor

It remains to be shown how the information about different types of packages (captured at the beginning of the process flow) can be combined with information about the multiple positions and conveyors and carried throughout the process so that the mappings used in Section 4.3 could make sense logically and allow specifications in Fig. 7 to work as intended.

This is the role of the DS model. It is fair to remark that designing a DS is an additional task introduced to the control project. Yet, it is generally simple and can be approached modu-

larly, as instances of events have different names and can be approached separately from each other.

To distinguish the event instances in the plant Ga, we consider that DSs are virtually positioned at strategic points along key steps of the process. Let us start by associating a DS with the conveyor D to capture the origin (A, B, or C) of each package and choose, based on this origin, which instances in $\Delta_1^{s_D} = \{s_{DA}, s_{DB}, s_{DC}\}$ and $\Delta_1^{f_D} = \{f_{DA}, f_{DB}, f_{DC}\}$ are really to be chosen when the signals s_D and f_D occur physically. This leads us to the six DS models, one for each instance in $\Delta_1^{s_D}$ and $\Delta_1^{f_D}$, shown in Fig. 8.

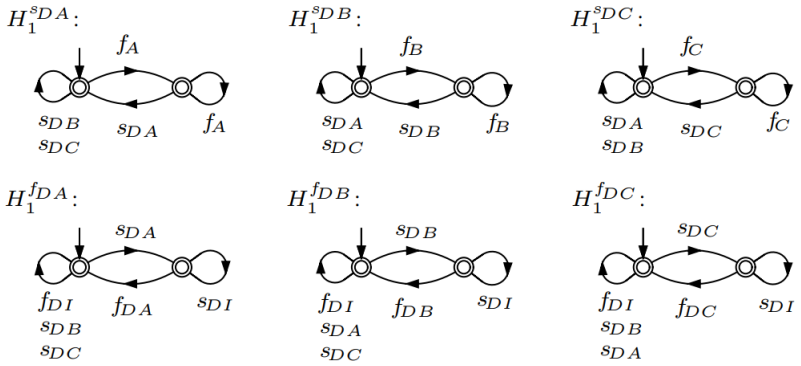

Figure 8 – Sensor $H_1^D = H_1^{s_{DI}} \parallel H_1^{f_{DI}}$, for $I = A,B,C$. It identifies the information about the type of package arriving from the conveyors A, B, and C (events f_A, f_B, or f_C) and carries these information through the conveyor D.

As for $\Delta_1^{s_E} = \{s_{EI}\}$ and $\Delta_1^{f_E} = \{f_{EI}\}$, for $\{s_{EI} \mid I = A, B, C\}$ besides carrying the type information through conveyor E they are also associated with the level 2 of refinements (because they recognize positions in addition to type). We let to display and explain their DS models later in Fig. 10.

After leaving conveyor D, packages are processed through conveyor E by reusing the previous distinctions that, so far, have carried only the memory of their type. In conveyor E, each type must be memorized in conjunction with the multiple positions each package can occupy. This will require two more groups of DSs, one for the type and the other for positions.

Consider starting with the distinctions of positions of E. Thus, a sensor model named H_2^{Pos} (with 2 denoting the second level of distinctions as in Section 4.3) is positioned in E to choose between the events $\Delta_2^{s_{EI}} = \{s_{EI1}, s_{EI2}, s_{EI3}\}$ and $\Delta_2^{f_{EI}} = \{f_{EI1}, f_{EI2}, f_{EI3}\}$, for $I = A,B,C$ again representing the type, and $1, 2, 3$ representing the 3 positions of conveyor E. Fig. 9 shows the result.

Finally, the group of sensors that provide information about the type, in addition to the positions, is denoted H_2^{Type} and it handles the logical sequence of transport of each type of packaging (A, B, or C) along the three positions of conveyor E, as shown in Fig. 10. The methodology used to construct the DSs is inspired by the one in (TEIXEIRA *et al.*, 2014).

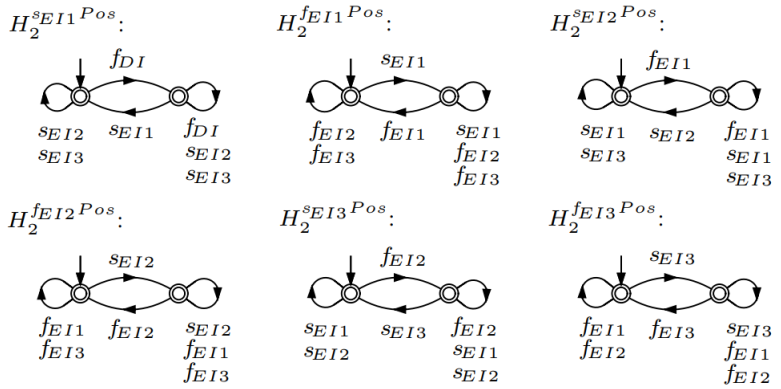

Figure 9 – Sensor $H_2^{Pos} = H_2^{sEIJ\,Pos} \parallel H_2^{fEIJ\,Pos}$, for $I = A,B,C$ and $J = 1,2,3$. Model for identifying which position of the conveyor E each package occupies.

4.6 Monolithic synthesis

The model $H = H_1^D \parallel H_2^{Pos} \parallel H_2^{Type}$ has 262144 states and 1769472 transitions. When composed with the plant $G\mathrm{a}$, it forms the plant $G_d = G\mathrm{a} \parallel H$, with 720896 states and 532544 transitions. The specification has 512 states, and the model for $K_d = G_d \parallel E_d$ has 369098752 states, which results in a supervisor S_d, that represents $\sup\mathcal{C}(K_d,G_d)$, with 293632 states and 1787264 transitions.

4.7 LMC with target-event-based approximations

The size of the models handled in monolithic synthesis reveals the difficulty faced both in obtaining and implementing the control system. This justifies efforts to reduce such complexity, and this section shows an alternative based on the LMC framework combined with approximations.

The first attempt to use approximations in LMC was in (TEIXEIRA; CURY; QUEIROZ, 2011), with later updates in (TEIXEIRA; CURY; QUEIROZ, 2018). Both works still consider the empirical selection of DS parts to include in synthesis. In the following, we consider the LMC approach (Section 2.2) combined with DSs (Section 2.3) and approximations (Section 2.4) with automated DS selection (Section 2.5).

The LMC with DSs works as the usual LMC. The control problem is seen as a set of instances, each one to be processed apart from the others and hopefully in a simpler manner. The difference is that an approximation is now set for each instance, and possible target events are handled. The result is a set of locally optimal and nonblocking supervisors whose combined behaviour is still to be verified, as in LMC, to make sure it does not block. When it does not, the result is optimal. The following steps can summarize the proposed modular synthesis process:

(a) *building suitable local approximations*: for each E_d^i, where $i \in Y = \{1,\cdots,m\}$, a local plant $\mathrm{G}_{\mathrm{a}i}^{loc'} = \Pi^{-1}(\mathrm{G}_i^{loc}) \parallel H_a$ is defined. Initially considering that $L(H_a) = \Delta^*$,

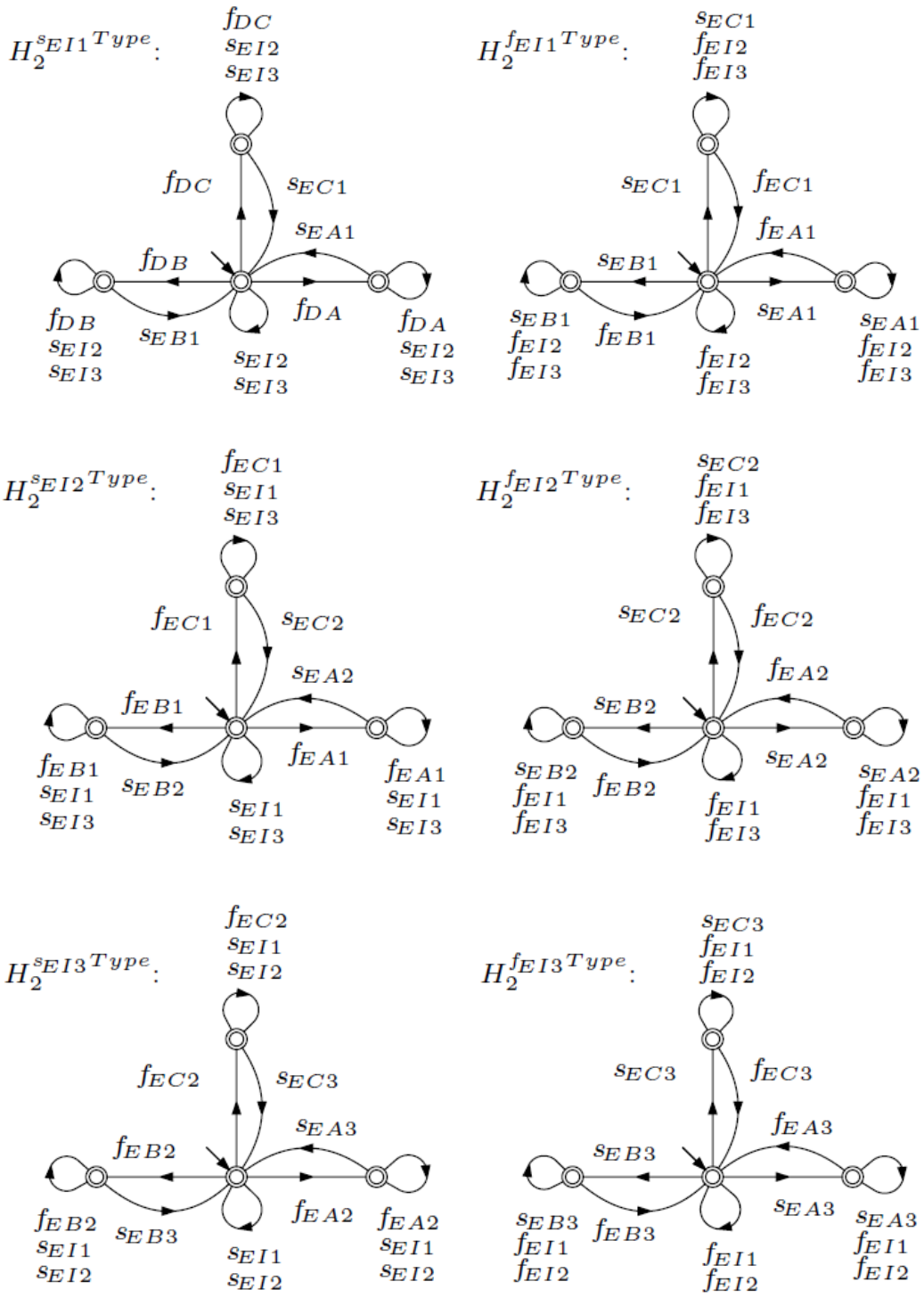

Figure 10 – Sensor $H_2^{Type} = H_2^{s_{EIJ}Type} \parallel H_2^{f_{EIJ}Type}$, for $I = A,B,C$ and $J = 1,2,3$. Model for identifying which type of package occupy each position on conveyor E.

i.e., H_a is assumed to be a version of the DS which does not distinguish any event. In practice, H_a can be modelled by a self-loop on a marked initial state, labelled with all events in Δ. It remains to know whether or not H_a has to be improved with some distinction. Considering the set $T^i \subseteq \Delta^u$ of target events for $\mathrm{G}_{\mathrm{a}i}^{loc'}$ due to disablements imposed by E_d^i (ROSA; TEIXEIRA; MALIK, 2018), a new version of H_a that handles these target events is obtained as $H_a^i = H_a \parallel H_{T^i}$, where $H_{T^i} = \|_{t \in T^i} H_t$ and H_t is an FSM that models a predictable distinguisher for each target event t identified for module i. Finally, $\mathrm{G}_{\mathrm{a}i}^{loc} = \mathrm{G}_{\mathrm{a}i}^{loc'} \| H_a^i$ represents a *suitable local approximation* (ROSA; TEIXEIRA; MALIK, 2018).

(b) *synthesizing local supervisors*: for each E_d^i and $\mathrm{G}_{\mathrm{a}i}^{loc}$, with $i \in Y = \{1, \cdots, m\}$, a local supervisor for plant $\mathrm{G}_{\mathrm{a}i}^{loc}$ is derived through $\sup\mathcal{C}(\mathrm{G}_{\mathrm{a}i}^{loc}, \mathrm{K}_{\mathrm{a}i}^{loc})$, where $\mathrm{K}_{\mathrm{a}i}^{loc} = \mathrm{G}_{\mathrm{a}i}^{loc} \parallel E_d^i$, and $\mathrm{S}_{\mathrm{a}i}^{loc}$ models the local supervisor.

Intuitively, if the set T^i, calculated for $\mathrm{G}_{\mathrm{a}i}^{loc}$ with respect to E_d^i, is empty, then one expects that $\mathrm{S}_{\mathrm{a}i}^{loc} \| H_d = \mathrm{S}_{\mathrm{d}i}^{loc}$, for $\mathrm{S}_{\mathrm{d}i}^{loc}$ modeling $\sup\mathcal{C}(\mathrm{G}_{\mathrm{d}i}^{loc}, \mathrm{K}_{\mathrm{d}i}^{loc})$ with $\mathrm{G}_{\mathrm{d}i}^{loc} = \mathrm{G}_{\mathrm{a}i}^{loc} \| H_d$. Furthermore, if the set $\{\mathrm{S}_{\mathrm{a}i}^{loc} \parallel H_d, i = 1, \cdots, m\}$ is non-conflicting, then the equality $S = \Pi(\|_{i=1}^{m} \mathrm{S}_{\mathrm{a}i}^{loc} \parallel H_d)$ is expected from the results in Section 2.2. To test this hypothesis, we revisited the example introduced in Section 4.6.

4.7.1 Local Modular synthesis for the example

For each module $i \in I$, Table 1 shows which plant components were used to form each $\mathrm{G}_{\mathrm{a}i}^{loc}$ and the number of states of $\mathrm{G}_{\mathrm{a}i}^{loc}$, $\mathrm{K}_{\mathrm{a}i}^{loc}$, and $\mathrm{S}_{\mathrm{a}i}^{loc}$. For this particular example, $T^i = \emptyset$, for each $i \in I$, meaning no target event has been in any module and synthesis can follow without using any additional DSs. In other words, when $T^i = \emptyset$, all the complexity inherent in the DSs models is removed from the synthesis process.

Table 1 – Synthesis setup for the LMC, DSs, and approximations applied to the example in Section 4.6.

LMC with DSs synthesis - states (transitions)					
$i \in I$	E_d^i	Plant components	$\mathrm{G}_{\mathrm{a}i}^{loc}$	$\mathrm{K}_{\mathrm{a}i}^{loc}$	$\mathrm{S}_{\mathrm{a}i}^{loc}$
1	2	G_a^1, G_a^4	2	4	4
2	2	G_a^2, G_a^5	2	4	4
3	2	G_a^3, G_a^6	2	4	4
4	2	$G_a^1, G_a^2, G_a^3, G_a^4, G_a^5, G_a^6, G_a^7$	16	32	32
5	2	G_a^4,G_a^5,G_a^6,G_a^7	16	32	32
6	2	G_a^6,G_a^7	4	8	6
7	2	G_a^8,G_a^9	4	8	6
8	2	G_a^9,G_a^{10},G_a^{11}	8	16	12
10	2	G_a^8,G_a^{10},G_a^{11}	8	16	12
$(\|_{i=1}^{10} \mathrm{S}_{\mathrm{a}i}^{loc}) \| H_d$					293632

Remark that each synthesis procedure iterates over FSMs $\mathrm{K}_{\mathrm{a}_i}^{loc}$ with at most 32 states. It leads to 9 supervisors that can be implemented separately and, in association with the DS model, impose on the plant the same control as the monolithic supervisor; in fact, their composition would also generate the same 293.632 states (see Section 4.6). In comparison, the monolithic synthesis with the full DSs model iterates over 369.098.752 states, while its version with approximations (the one that equivalently does not add any DS model to the synthesis) iterates over 11.520 states. They both lead to the same final supervision system with 293.632 states.

The Supremica tool (AKESSON *et al.*, 2019) was used for design, synchronization, simulation, and synthesis. However, it has limited resources for modelling translation and automated code generation. In response to this limitation, we developed a dedicated code generation tool (POSSATO, 2023) that complements Supremica's output with implementation resources.

5 A CODE GENERATION TOOL

The *DESPythonMaker* is a tool programmed in Python that receives a Supremica file as input and exports a directory with a project in Python, from where the integration with artificial intelligence resources flows naturally. The tool supports monolithic and modular synthesis with or without DSs. The directory with the generated code is organized to allow the quick implementation of the project and the easy replacement of supervisors in case of modelling updates.

Figure 11 shows the flow involving the code generation. DESPythonMaker must be called from the command line, setting the path for the Supremica file to be processed and the name of the output file. DESPythonMaker then scans the file, identifies the controller, and builds the executable output project. Moving the files to a specific structure may be necessary, depending on the target platform, which can be automated by the script we provide in (POSSATO, 2023).

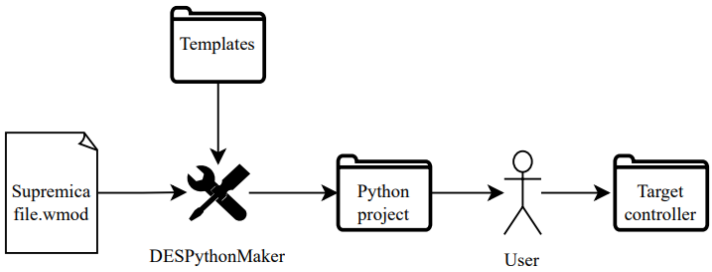

Figure 11 – Conceptual flow of the DESPythonMaker tool.

The hierarchy proposed for the tool is inspired by (QUEIROZ; CURY, 2002) and adapted as in Figure 12.

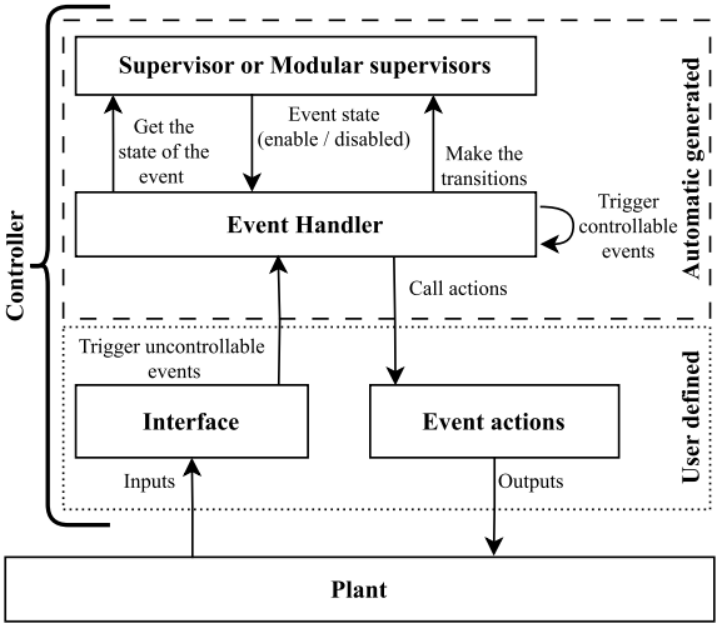

Figure 12 – Architecture of the code generation tool.

The architecture is divided into two main blocks: one describing the automatic code generation operations and another with user-defined settings. The first block includes two main components: the structure of the *supervisors* and the *event handler*. The second block includes the setup area for the uncontrollable events to be transmitted to the event handler, and the actions associated with the events, usually controllable events (POSSATO, 2023).

5.1 The Supervisor Structure

Supervisors are represented through linked lists, with classes representing events, states, and transitions. These interconnected structures allow access and navigation through the state space. The classes defining events, states, and transitions have methods for performing operations on the supervisors, such as adding a callback function to each event and checking whether an event is enabled. Each event can have an associated callback function that is executed whenever the event is triggered (POSSATO, 2023).

The supervisor is also represented by a class that points to an initial state, the current state, the next state, and an alphabet. The alphabet is a list of all the events of the supervisor. A state has a list of transitions. A transition has the target state to be reached with an associated event. The event has a type (controllable or uncontrollable) and an associated action calling responsible for triggering the real action in the plant. Figure 13 illustrates these relationships (POSSATO, 2023).

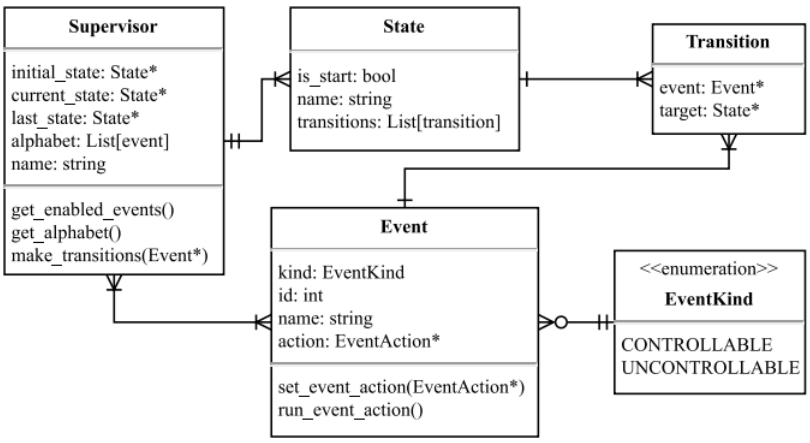

Figure 13 – Entity Relationship description of a supervisor.

It is important to note that the events are global and not linked directly to a supervisor. This is why there is only one event structure, even using the modular approach. The event handler is proposed to check whether an event is enabled and make the transitions in the automata accordingly of the distinguishing sensors and supervisors.

5.2 The Event Handler

The event handler manages the events of the controller. For its correct operation, every event must be submitted to the event handler through the "trigger_event" function. The event handling checks whether supervisors enable the event. If so, it is triggered, meaning making a transition in the FSM, calling the function associated with the event action, and managing the controllable events enabled in the plant. Figure 14 presents a flow chart to explain the operation of the event handler, including its interaction with the plant and supervisor.

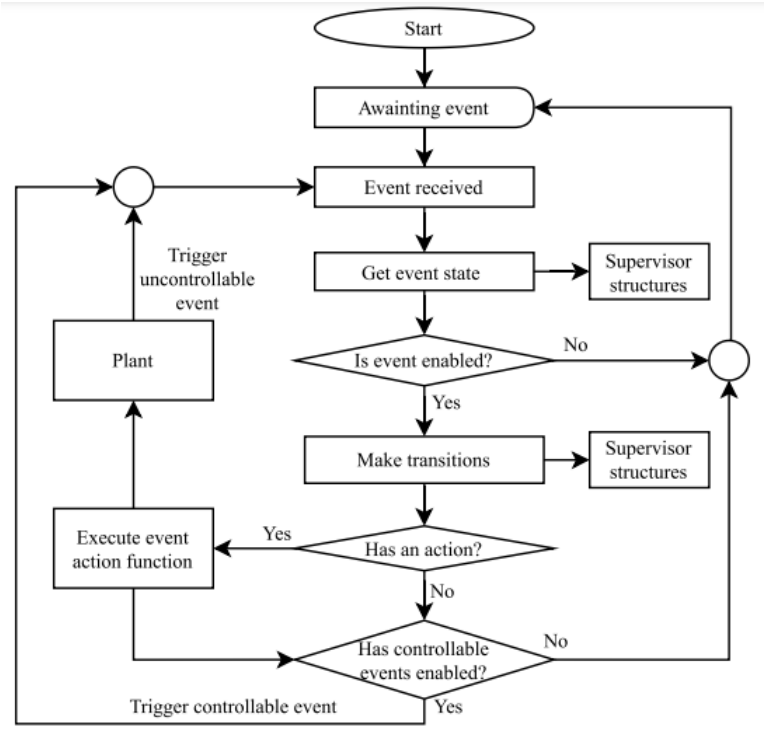

Figure 14 – Flowchart for the event handler operation.

The effect of the modular control approach is incorporated into the tool in the event handler structure. It is materialized by a list of supervisors that is consulted by the event handler to check whether all the supervisors enable each event. An event is only enabled in the plant under control when this checking is true, which materializes the idea of event disabling for the conjunction of modular controllers (POSSATO, 2023).

5.3 The User Area

The user area includes the functions for interfacing with the physical plant. In the *Interface* block, the plant signals are read, converted into uncontrollable events, and transmitted to the event handler. In the *Event actions* block, each controllable event is associated with a function. This function outlines a set of directives that must be carried out in the physical plant, whenever the corresponding event is triggered in the supervisor. For example, in process broiler

meat (Section 4), the event s_A is linked with a function that specifies a set of low-level commands that enables the motor of conveyor A for a given time and subsequently disables it.

6 CASE STUDY IMPLEMENTATION

In this section, we implement the solution for the example in Section 4 by using the automated code generation tool presented in Section 5. Then, we simulate the DES of the example under the influence of the implemented controller. The Factory I/O[1] simulator was used to replace the real plant and the real controller. This tool reproduces by software the same conditions as a real plant in terms of devices, hardware, communication channels and protocols, such as Modbus TCP used here, and many other resources, while additionally allowing test scenarios.

Figure15 summarizes the main elements involving the steps that culminate in the final implementation.

Figure 15 – Implementation architecture for the example.

The main elements are:

- Modbus communication between Factory IO and the main program.
- The main program used to associate the digital inputs and outputs of Factory IO with the action functions created to handle events.
- The custom event handler, which handles uncontrollable events and executes controllable events in the FSM of the supervisor and DSs.
- The supervisor and DSs obtained by converting the output from Supremica.

6.1 Modbus Interface

The Modbus interface is responsible for communication with the simulator. The digital inputs corresponding to sensors A, B, and C are constantly read, and a function is called to process the value. Listing 6.1 shows a code sketch with the function for sensor A.

In Listing 6.1, whenever the signal of sensor A is evaluated to True, the program executes the action function associated with that digital input, which triggers the event s_A to the supervisor through *trigger_event(Events['SA'])*. The same procedure is followed for sensors B and C.

Listing 6.1 – Function called whenever the interface detects a change in the digital input corresponding to sensor A.

```
1 def sensor_A_action(value):
2     if(value == True):
```

[1] https://factoryio.com

```
3           # Trigger the corresponding event.
4           trigger_event(Events['SA'])
```

The Modbus interface also provides a class for creating digital output objects with methods to turn the corresponding output in Factory IO on and off.

6.2 Main Program

The main program connects the Modbus interface with the event handler. It defines functions to pass uncontrollable events from the plant to the event handler and functions associated with controllable events.

Listing 6.2 – Action function to start motor A.

```
1  def motor_iA_action(event):
2      # Send a command via Modbus to activate the motor.
3      motor_A.on()
4
5      # starts a timer to start the motor of conveyor D
6      # which removes the box from conveyor A
7      Timer(1.5/FACTORY_IO_TIME_SCALE, motor_D_iA_on).start()
8
9      # starts a timer to turn off the motor $A$
10     Timer(2/FACTORY_IO_TIME_SCALE, motor_A.off).start()
```

Listing 6.2 presents the code sketch for the action function associated with the output event to turn on and off motor A, *motor_iA_action*.

6.3 Custom Event Handler

In summary of how the implementation works, when the physical sensors are triggered in Factory IO, this information is passed to the Event Handler through the Modbus Interface. The event handler checks which events are enabled in the state machines of both the Supervisor and the Distinguishers simultaneously, and then executes these events in the state machines and performs the action functions associated with these events.

The event handler can be implemented in a distributed way through IoT modules that communicate with the plant to inform contexts, and they may be interdependent among each other.

For this purpose, a list was created containing the modular supervisors and distinguishers. In other words, each distinguisher can be housed in different hardware, provided there is adequate communication and synchronization with the main control system. Each time events are received by the event handler, they are mandatorily checked across all supervisor modules

and all distinguishers to ascertain if these events are enabled simultaneously. Only under this condition will the state machines and action functions be executed.

The event handler module of the DESPythonMaker had to be extended in this paper compared to its seminal release presented in (POSSATO *et al.*, 2023). The modification aims to accommodate the idea that DSs perform a complementary role to the plant.

To exemplify the extension, consider the case of conveyor E, which has three context positions associated with the same events of turning its motor on and off. In the standard release of the tool, the event handler was conceived to execute all controllable events whenever they are allowed in the plant under control. However, when it comes to interdependent distinctions, such as in the example of conveyor E, a single activation of its motor implies shifting all other distinctions that depend on it before another activation may occur.

This notion extends the intuitive idea of predictability, which says that a single distinction is possible for each event in the real plant. In the case of interdependent DSs, this can be rewritten as “a single distinction is possible at each level, for each event in the real plant". Ensuring the shifting between each consecutive level to reach this effect is therefore essential when implementing this kind of solution, and we consider this an implementation argument.

A list of blocked controllable events was created to achieve this goal, which the event handler will not automatically execute. For the particular example, the events within this list include s_{EA2}, s_{EB2}, s_{EC2}, s_{EA3}, s_{EB3}, and s_{EC3}. Their selection is manual; we do not see how this can be automated, as it is application-dependent. In other words, only the instances of events from the first position of conveyor E (s_{EA1}, s_{EB1}, and s_{EC1}) are associated with the action function that sends the signal to Factory IO to turn the conveyor E on and off. The instances related to positions 2 and 3 serve only for context shifting. They are executed in the user area, triggered orderly according to context transfer, i.e., s_{EA3}, s_{EB3}, s_{EC3}, s_{EA2}, s_{EB2}, and s_{EC2}, every time an event f_{EIX} occurs, for $I = A, B, C$ and $X = 1, 2, 3$.

The extended event handler implementation also includes a minor feature intended to add justice to the removal of packages from the conveyors, preventing them from not being served or staying idle. For that, a FIFO (first-in-first-out) algorithm is set to select the oldest s_A, s_B, or s_C event observed.

A video complement demonstrating the usability of the approach over the case study presented in this paper is available in (VALENTINI, 2023).

7 CONCLUSION

This paper presented the theoretical foundation and an example of how to design, synthesize, and implement modular control systems for DES subject to the use of multiple interdependent distinguishing sensors.

Our approach joins previous efforts in the literature to tackle two main limitations faced when implementing industrial-scale controllers: design and processing complexities. Here, design limitations are tackled by extending previous ideas about distinguishing sensors, additionally considering that sometimes they can be interdependent. As for the processing limitations, we mitigate them by adopting modular strategies for control synthesis integrated with the use of abstraction-based models that approximate the DES behaviour so that synthesis cost is reduced while its outcome is still optimal.

For future work, we expected more in-depth investigations into the modelling of interdependent distinguishing sensors. It has been observed that progressive levels of distinctions allow distinguishers to assume different faces at distinct phases, and the lack of a modelling procedure seems to be a gap. We also believe that the IoT-based implementation still requires further investigation. One possibility is to transform context-shifting distinguishers into dedicated hardware modules capable of applying black-box edge computing to provide cleaner and more meaningful events to the plant.

BIBLIOGRAPHY

ÅKESSON, K.; FLORDAL, H.; FABIAN, M. Exploiting modularity for synthesis and verification of supervisors. **IFAC Proceedings Volumes**, v. 35, n. 1, p. 175 – 180, 2002.

AFONSO, P. de A.; FERREIRA, P. R. Autonomous robot navigation in crowd. *In*: IEEE. **Latin American Robotics Symposium (LARS)**. [*S.l.*], 2022. p. 139–144.

AKESSON, K. *et al.* **Supremica**. [*S.l.*], 2019. Disponível em: http://www.supremica.org/.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to Discrete Event Systems**. 3. ed. Switzerland: Springer Nature, 2021.

CURY, J. E. *et al.* Supervisory control of discrete event systems with distinguishers. **Automatica**, Elsevier, v. 56, p. 93–104, 2015.

LUCA, A. D.; ORIOLO, G.; GIORDANO, P. R. Image-based visual servoing schemes for nonholonomic mobile manipulators. **Robotica**, Cambridge University Press, v. 25, n. 2, p. 131–145, 2007.

MALIK, R.; TEIXEIRA, M. Modular supervisor synthesis for extended finite-state machines subject to controllability. *In*: **Workshop on Discrete Event Systems (WODES'16)**. [*S.l.*: *s.n.*], 2016. p. 91–96.

MALIK, R.; TEIXEIRA, M. Optimal modular control of discrete event systems with distinguishers and approximations. **Discrete Event Dynamic Systems**, v. 1, p. 1–12, 2021.

MOHAJERANI, S.; MALIK, R.; FABIAN, M. Compositional synthesis of supervisors in the form of state machines and state maps. **Automatica**, v. 76, p. 277 – 281, 2017.

POSSATO, T. **Automated code generator from Supremica to Python**. 2023. Disponível em: bit.ly/3saIY99.

POSSATO, T. *et al.* Automated code generation for DES controllers modeled as Finite State Machines. *In*: SBC. **26TH Brazilian Symposium on Formal Methods (SBMF)**. [*S.l.*], 2023. p. 1–10.

QUEIROZ, M. de; CURY, J. Synthesis and implementation of local modular supervisory control for a manufacturing cell. *In*: **Int. Workshop on Discrete Event Systems**. [*S.l.*]: IFAC, 2002. p. 377–382.

QUEIROZ, M. H. D.; CURY, J. E. R. Modular supervisory control of large scale discrete event systems. *In*: **Discrete Event Systems: Analysis and Control (WODES)**. [*S.l.*]: Kluwer Academic, 2000. p. 103–110.

QUEIROZ, M. H. D.; CURY, J. E. R. Modular multitasking supervisory control of composite discrete-event systems. **16th IFAC World Congress**, Prague, Czech Republic, 2005.

RAMADGE, P. J. Some tractable supervisory control problems for discrete-event systems modeled by buchi automata. **IEEE Transactions on Automatic Control**, IEEE, v. 34, n. 1, p. 10–19, 1989.

ROSA, M. *et al.* Efficient implementation of distinguished controllers for discrete-event systems. **IFAC-PapersOnLine**, Elsevier, v. 50, n. 1, p. 1187–1192, 2017.

ROSA, M.; TEIXEIRA, M.; MALIK, R. Exploiting approximations in supervisory control with distinguishers. *In*: **14th International Workshop on Discrete Event Systems**. [*S.l.*]: IFAC, 2018. p. 24–29.

SILVA, A. L.; RIBEIRO, R.; TEIXEIRA, M. Modeling and control of flexible context-dependent manufacturing systems. **Information Sciences**, v. 421, p. 1 – 14, 2017.

TEIXEIRA, M.; CURY, J. E. R.; QUEIROZ, M. H. de. Local modular supervisory control of des with distinguishers. *In*: **IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'11**. Toulouse, France: [*s.n.*], 2011. p. 1–8.

TEIXEIRA, M.; CURY, J. E. R.; QUEIROZ, M. H. de. Exploiting distinguishers in local modular control of discrete-event systems. **IEEE Transactions on Automation Science and Engineering**, v. 15, n. 3, p. 1431–1437, July 2018.

TEIXEIRA, M. *et al.* Supervisory control of des with extended finite-state machines and variable abstraction. **IEEE Transactions on Automatic Control**, IEEE, v. 60, n. 1, p. 118–129, 2014.

VALENTINI, J. *et al.* Interdependent distinguishing sensors applied to dispatch control in the poultry industry. *In*: IEEE. **IEEE International Conference on Industry Applications (INDUSCON)**. [*S.l.*], 2023. p. 1=10.

VALENTINI, J. H. **Interdependent distinguishing sensors application example**. 2023. Disponível em: https://bit.ly/464sA8y.

WONHAM, W.; RAMADGE, P. Modular supervisory control of discrete-event systems. **Mathematics of Control, Signals, and Systems (MCSS)**, Springer London, v. 1, p. 13–30, 1988.