International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 4
ISSN: 2088-5334

PUTRACOM: A Formalism of a Novel Component Model

Faranak Nejati[a,*], Ng Keng Yap[a], Abdul Azim Abd Ghani[a], Azmi Jaffar[a]

[a]*Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang 43400, Selangor, Malaysia*
E-mail: [*]faranak.nejati@student.upm.edu.my

***Abstract*— The composition mechanisms and interactions of current component models are mostly base on port or function calls from other components. However, in both styles, the number of interactions that depend on the number of ports and method calls may increase dramatically. Hence, to avoid such complexity of composing components and coordination of the interaction among them, a component model and policy to provide a separation between the components and coordinating is needed. This study presents a formal specification of a novel component model for discrete-event and non-blocking component-based systems called PUTRACOM. A new component model, named PUTRACOM is presented in this paper. PUTRACOM supports to develop concurrent software systems with discrete-event. PUTRACOM defines components by two essential features; they are the Observer/Observable unit and a computation unit. These two features allow a component to have fixed behavior without any dependency on other components. The components can be composed using a well-defined set of connectors. PUTRACOM has been formally defined based on the well-defined and sound methods like CSP and RTSs. PUTRACOM provides a way to construct components and coordinate them with a well-founded mechanism. The model defines a set of exogenous connectors and an observer/observable unit to encapsulate components and coordination. In order to illustrate the way of component composition in the proposed model, an example of the control system of a refrigerator is used. Moreover, to evaluate its applicability, the example has been implemented in Colored Petri Net (CPN) tools.**

***Keywords*— component-based software development; component model; encapsulation; computation; exogenous connectors.**

I. Introduction

Component-based software development (CBSD) is a new paradigm to tame the complexity of constructing large software systems. However, constructing the software components, synthesizing them together, and coordinating them are challenging tasks. It is because of complex composition styles conferred in the current component models. Their way of composing component and interaction among them are mostly based on port calling a function from other components (for example, ADLs). The complexity becomes worse when computation and coordination are not separated, and some couplings exist between the components in the concurrent component models. To address this problem, a component model that supports the reliable separation of computation and coordination is required.

In this paper, a concurrent component model called PUTRACOM that can mitigate the complexity of the component-based model by encapsulating computation and coordination among composed components is presented formally. The first contribution is the concept of the Observer/Observable Unit (OOU) in the components to encapsulate computations. OOU notifies the connectors, then the components are free from any direct message passing or invocation, which leads to encapsulation. Encapsulation prevents direct intervention among the components. Therefore, coupling in the model will be eliminated. OOU also avoids having multiple ports like what is presented in the typical component models to reduce complexity.

Second, a new set of exogenous connectors to encapsulate coordination is proposed. The considerable characteristic of our exogenous connectors is that the components never be engaged in control. In this way, computation and coordination will be separated. The novelty of the new exogenous connectors is the observation of OOUs by their subscribed components. By observing OOUs, connectors handle coordination all over the system without involving the computation unit of components.

Third, our connectors compose components synchronously, asynchronously, sequentially, conditionally, and iteratively. Finally, the behavior of the components and connectors are defined formally. This research defines components using Reactive Transition systems (RTS) [1], [2]. The composition and interactions between components are expressed in Communication Sequential Processes (CSP) [3], [4]. The proposed model has been evaluated to prove its applicability to the real-life systems in Colored Petri Net Tools (CPN) [5].

PUTRACOM is based on exogenous connectors. Exogenous connectors control the interaction between components outside of the components. The proposed component model in this study is unlike ADL likes component model [6]–[9]. ProCom also is used in real-life systems [10]. One component model with this kind of connectors is X-MAN [11]–[13]. X-MAN component model has some similar aspects to the proposed model in this work. It encapsulates computation and control and let the components to be decoupled. X-MAN is an exogenous based component model. Components never call methods or functions in other components. Connectors construct composed components. Connectors in X-MAN encapsulates control between components and let the component and composite component to be decoupled [14]. However, their connectors encapsulate only sequential controls. The underlying philosophy behind X-MAN is a proper method for our goal. This study enriched X-MAN by extending this model to support concurrency.

To specify the model formally, it is vital to choose a proper formalism which can define all the essential properties of components, connectors, composition mechanism, and interactions among them. Communication Sequential Processes (CSP) [3] is a well-defined parallel computation formalism to define processes running in parallel. The processes communicate using multiple operators such as sequential composition, broadcasting, parallel composition, iteration, and conditional composition. The connectors in PUTRACOM are also defined based on these operators. However, the message passing communication model in CSP is not supported in PUTRACOM. Processes in CSP are defined by Label Transition Systems (LTSs). LTSs express the behavior of processes and semantic of all composition operators. However, in LTS, input, output, and internal events are not distinguished explicitly. In this paper, an improved kind of LTS called Reactive Transition Systems (RTSs) had been adopted [1]. RTS models the internal behavior of components and separates it from input and output behavior.

II. Material and Method

This section aims at defining atomic components. All components consist of a two different kinds of units called CU and (OOU). As shown in Fig. 1, CU (Computation Unit) encapsulates computation that it will never call other components. Input/output events with their corresponding data are manifested as a multiset in OOU (Observer/Observable Unit). The computing environment and the connector that the component subscribed can observe and use the information in OOU.

Definition 1. The CU of an atomic component is an RTS (Reactive Transition System), CU = (s_0, S, E, V, Δ).

S is a set of states in each RTS. E is a set of events which comprises three kinds of events: Output events E^O, Input events E^I, and hidden events E^H. E^I and E^O are considered as a set of observable events E^{Obs} which can be distinguished by "?" and "!" respectively. Each event has its corresponding data, which are presented by V. Δ is a set of transitions from one state to another. The transitions are labeled events and their corresponding data regardless of the kind of event. If the transition is admissible, then the control will be passed to the next state. An atomic component in detail is presented in Fig. 1.

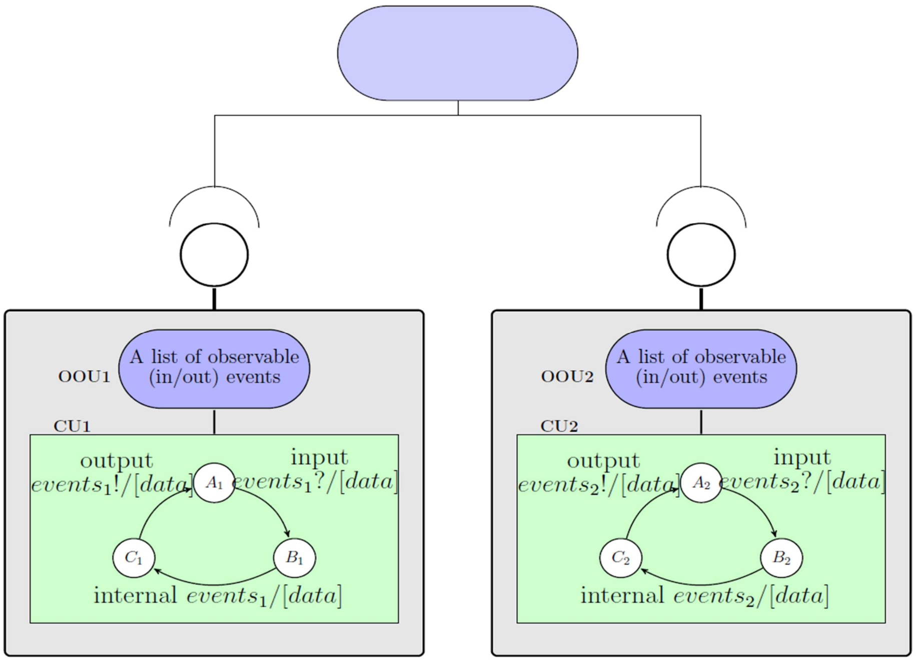
OOU1
A list of observable (in/out) events
CU1
output $events_1!/[data]$
input $events_1?/[data]$
A_1
B_1
C_1
internal $events_1/[data]$
OOU2
A list of observable (in/out) events
CU2
output $events_2!/[data]$
input $events_2?/[data]$
A_2
B_2
C_2
internal $events_2/[data]$

Fig. 1 Components with details in PUTRACOM

A. Composition of Atomic Components

To compose components, the proposed model is enriched by connectors. Connectors are composition operators to compose and control the communication among components. In this section, it has expressed the meaning of these composition operators mathematically using CSP. Synchronization constraints, coordination, and communication between components are defined in connectors.

Connectors observe the OOUs of their subscribed components and manifest the output events of a component to input events of other components. The following definition specifies a connector [15].

Definition 2. Let Γ be a connector to compose components; then the connector Γ is a tuple $\Gamma = (L, T, Sub, \prod, G)$,

L indicates the set lines that are used to connect a component to the connector, and through those lines, they can communicate with the components. As the proposed model support multiple types of synchronization, T is used to define the types of connectors. These types are $sync, async, con, seq, and\ itr$;

Each component that is connected to the connector through Lis called subscriber. $Sub = \{b_0, b_1, b_2, \ldots, b_n\}$ is the set of the subscriber to a given connector.

The set of interactions among the component's id indicated by $\prod .G = \{g_0, g_1, \ldots, g_n\}$ is a set of constraints.

1) Type of Connectors: The types of connectors indicate the synchronization types are that enforced upon components and interaction. The connector's types defined in PUTRACOM are elaborate in this section.

- *Sync connector*: Two components C_1 and C_2 can be composed and coordinated synchronously by a $sync$ connector shows by Γ_{sync}. When an event occurs, if the corresponding G is satisfied, all the components subscribed to the Γ_{sync} will change their state. In case of any component is not ready, the rest of them will be blocked.
- Fig. **2** represents a $sync$ connector.

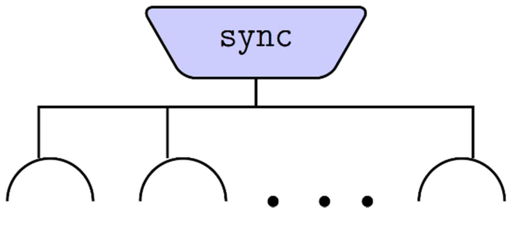

Fig. 2 ***sync*** connector in PUTRACOM

- *async connector:* In contrast to *sync* connector, an *async* connector may act independently. It means the concurrent behavior of components *C₁ and C₂* is like running both in parallel, which has interleaving traces. Fig. 3 indicates an *async* connector. The following definition specifies *sync* [15].

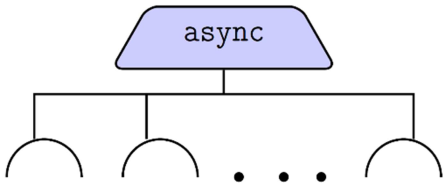

Fig. 3 ***async*** connector in PUTRACOM

- *Other Connectors:* In the case that there is a connector to choose occurring a specific event, it has been used conditional connectors. Another type is sequential connectors run components sequentially. The first component is waiting until the other is terminated. The last kind of connector is an iterative connector, which runs a component sequentially in infinite times.

2) Interactions: Interactions are the communications between the components. All the interactions are under the control of the connectors, and no components can intervene. Depends on the type of each connector, it coordinates the events from the computing environment or other OOU of components. This coordination is totally based on the synchronization constraints. Synchronization constraints must be defined before in G. According to the types of connectors, the type of interactions may differ.

3) Composition: In the composition of two components, the precondition is encapsulating computation. It is because and the function call is forbidden in PUTRACOM. The following definition defines the composition of components in PUTRACOM.

Let two C_1 and C_2 which is defined by RTSs. Then, a composition of these two components could be defined $N = (r_0, R, E_{(c_1 \otimes c_2)}, V, \Delta)$.

R indicates a set of components (or RTSs) includes two components C_1 and C_2.

E indicates a set of Hidden and Observable events of both components C_1 and C_2. The Hidden Events are:

$$E^H_{C_1 \otimes C_2} = \{e \mid e \in E^O_{C_i} \wedge e \in E^I_{C_j}, \forall i, j \in \{1,2\}\};$$

Observable events are $E^{obs}_{C_1 \otimes C_2} = E^I_{C_1 \otimes C_2} \cup E^O_{C_1 \otimes C_2}$ where $E^I_{C_1 \otimes C_2} = (E^I_{C_1} \cup E^I_{C_2}) / (E^H)$ an $E^O_{C_1 \otimes C_2} = E^O_{C_1} \cup E^O_{C_2}$;

The set of variables and transition in the composition of two components are

$V = \{v_1, v_2, \ldots, v_n\}$ and $\Delta \subseteq R \times \prod \times R$ respectively.

III. Result and Discussion

In this section, an example has been defined [11] and it is used to demonstrate a control system for a refrigerator. The refrigerator includes a cooler, a temperature sensor, and a switch component to turn on and off the cooler. A component has been specified for controlling the cooler and temperature sensor [11]. However, this study does not use such components because our connectors are able to control and coordinate all these components accordingly. The cooler is a component for cooling in a refrigerator. When the switch component is on, the input event $start?$ in the cooling component occurs and t_2 triggers idle state to the cooling state. When the temperature's variable $temp$ (which control by the sensor component) drops below the threshold X, then the event $stop$? occurs and firing the transition t_3 tigers the cooler component from $cooling$ state to $idle$. The general behavior of the cooler component is:

$$cooler \xrightarrow{e} cooler' \Rightarrow e = \{start?, stop?\} \Rightarrow$$

$$idle \xrightarrow{start?} Coolling, coolling \xrightarrow{stop?} idle$$

A sensor will check the temperature of the refrigerator as in Figure 4 below. The sensor component has four states observing, decreasing, increasing, and normal. When $temp$ drops below the low threshold X, then event $high$? will happen and transition t_4 triggers to $increasing$ state. Then $temp$ will slowly increase until the threshold Y is met. Currently, the cooler component remains in $idle$ state. If $temp$ rises and exceeds the high threshold Y, the $decreasing$! the event will occur, and the cooler component starts to cool again.

Whenever the event OFF? from the $switch$ component occurs, t_6 triggers a cooling state to idle. In the refrigerator controller, switch and cooler run asynchronously. They can react whenever admissible events occur. To compose the two components, an $async$ connector is used. Let Com_1 is a composition of switch and cooler, $switch \mid\mid\mid cooler$, then the behavior of them is defined below:

$$Com_1 \xrightarrow[\Gamma_{async}]{} Com_1' = \{start!start?$$

$$=\{on, cooling\} \cup stop!stop?$$

$$=\{off, idle\}\}$$

Com_1 is the synchronously composed with temperature component. Let Com_2 is the synchronous composition of Com_1 and temperature, $Com_1 \mid\mid temperature$, then the behavior of Com_2 is specified bellow:

{start! & start? & high? = {on, idle, decreasing}

∪ stop! & stop? = {off,idle}

∪ start! & start? & low? = {on, idle, increasing}

∪ start! & start? & start! = {on, cooling, observing}

∪ stop! & stop? & {high?, low?, start!} = {on, idle}}

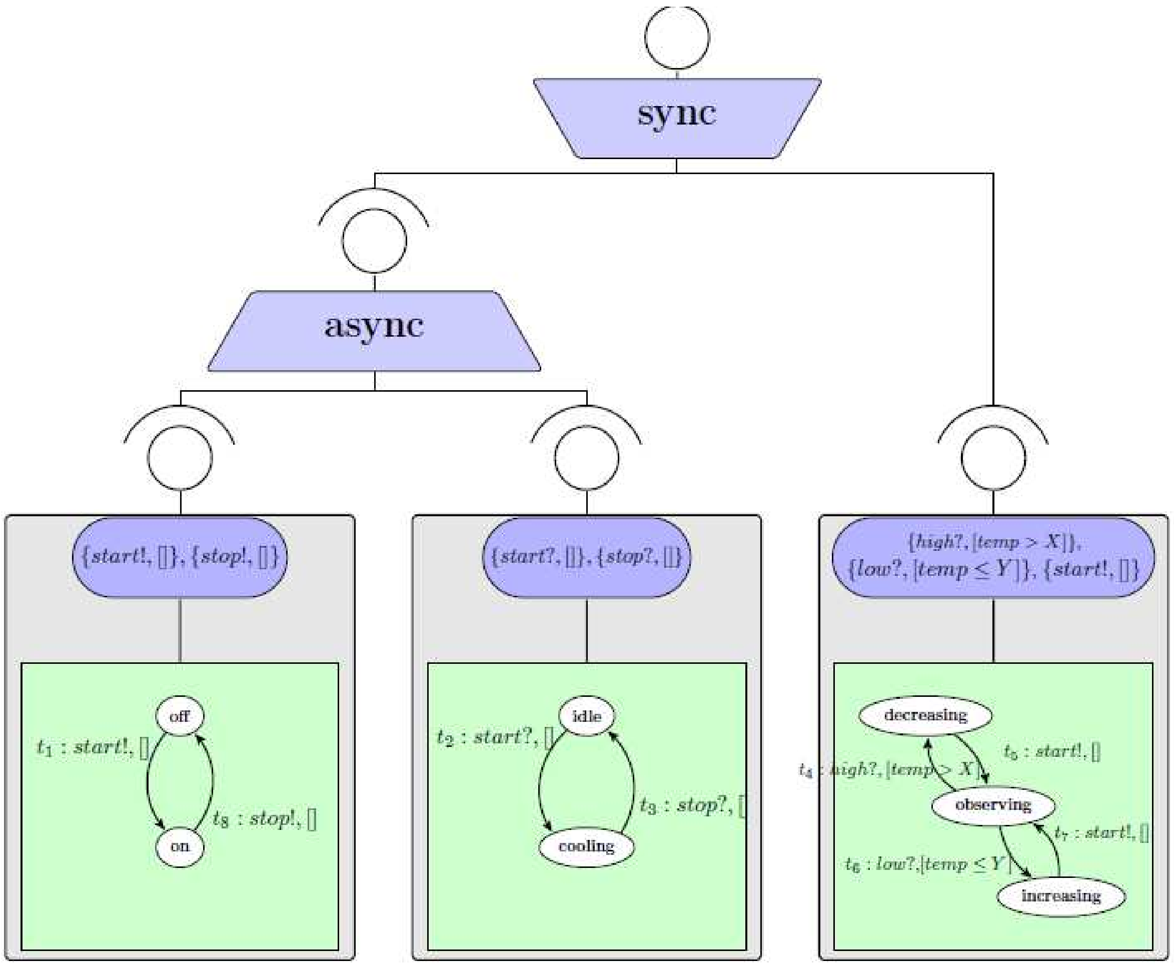

Fig. 4 A refrigerator controller by PUTRACOM

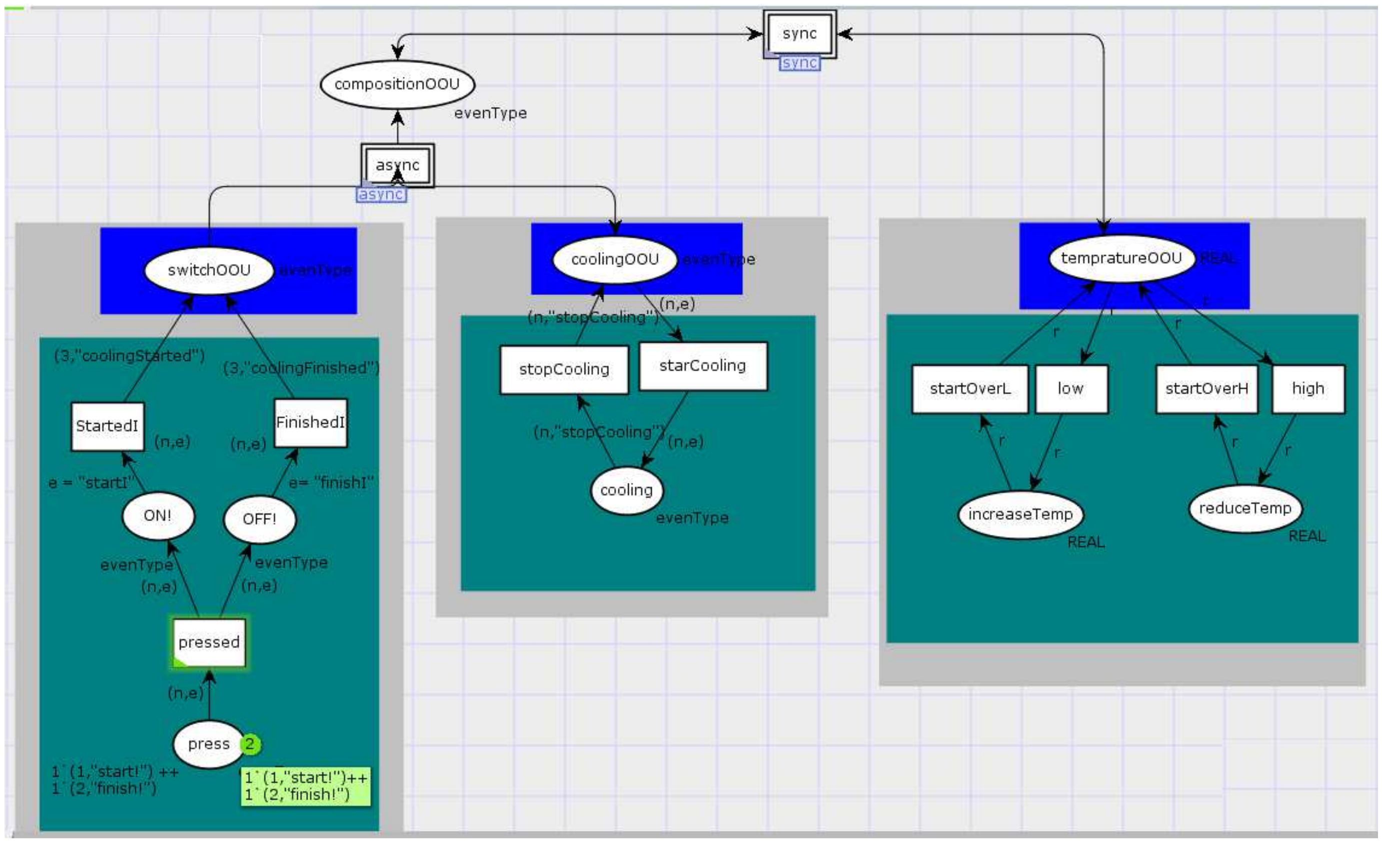

Fig. 5 An example of PUTRACOM modeled in CPN tools

The cooling system of a refrigerator is modeled based on PUTRACOM component model in Colored Petri Net (CPN) tools, Fig. 5. CPN Tools is a tool for modeling systems and exhibiting multiple kinds of synchronization. It is based on the Petri net to simulate concurrent or multiple processes systems [9]. It provides a platform to analyze the PUTRACOM and prove its applicability.

The CPN version of PUTRACOM indicates that encapsulating computational units to be decoupled from the other components, utilizing OOU to leave CU totally encapsulated, having exogenous connectors with multiple kinds of synchronization, and finally fully encapsulate control are applicable. In other words, the underlying consents of PUTRACOM are applicable.

IV. Conclusion

A new component model, named PUTRACOM, is presented in this paper. PUTRACOM supports to develop concurrent software systems with discrete-event. PUTRACOM defines components by two important features: the Observer/Observable unit and a computation unit. These two features allow a component to have fixed behavior without any dependency on other components. The components can be composed using a well-defined set of connectors.

Each component is enriched by a novel unit called OOU for notifying the connectors then the components are free from any direct message passing or invocation. Moreover, there are no multiple ports or method calls. Therefore, the complexity will be reduced. Exogenous connectors can handle coordination all over the system without evolving components. Its applicability by utilizing an example and model it in the CPN tools is demonstrated.

PUTRACOM has been formally defined based on the well-defined and sound methods like CSP and RTSs. We intend to develop a prototype tool for it. Moreover, PUTRACOM has the potential to construct incrementally. Constructing systems bit by bit reduce the complexity of the system. This is the future work that aimed to be investigated. It may also help in the verification of component-based systems [16].

Acknowledgment

The completion of this project could not have been accomplished without the support of the Ministry of Higher Education Malaysia and Universiti Putra Malaysia for their generous grant “Project: UPM/700-2/1/FRGS/08-01-15-1719FR”.

References

[1] Y. Jin, “Compositional verification of component-based heterogeneous systems,” University of Adelaide, 2004.
[2] B. Igried and A. Setzer, "Programming with monadic CSP-style processes in dependent type theory," Proceedings of the 1st International Workshop on Type-Driven Development. ACM, 2016.
[3] A. W. Roscoe, S. D. Hoare, and C. A. R. Hoare, A theory of communicating sequential processes. Oxford University Computing Laboratory, Programming Research Group, 1981.
[4] B. Igried and A. Setzer, "Programming with monadic CSP-style processes in dependent type theory," Proceedings of the 1st International Workshop on Type-Driven Development. ACM, 2016.
[5] K. Jensen, Coloured Petri nets: basic concepts, analysis methods and practical use, Vol. 1. Springer Science & Business Media, 2013.
[6] J. El Hachem, et al. "Model driven software security architecture of systems-of-systems." 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2016.

[7] F. Oquendo, J. Leite and T. Batista, "Specifying architecture behavior with SysADL." 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE, 2016.

[8] A. Butting, et al. "Systematic language extension mechanisms for the MontiArc architecture description language." European Conference on Modelling Foundations and Applications. Springer, Cham, 2017.

[9] M. D. Sanctis, et al. "A model-driven approach to catch performance antipatterns in ADL specifications." Information and Software Technology 83 (2017).

[10] X. Jiang, et al. "ProCom: designing a mobile and wearable system to support proximity awareness for people with autism," Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct. ACM, 2016.

[11] K. K. Lau, K. Ng, T. Rana, and C. M. Tran, "Incremental construction of component-based systems," in Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering, 2012.

[12] K. Lau, C. M. T. X-MAN tool set, 2016.

[13] K. Lau. An Introduction to Component-based Software Development. Vol. 3., 2017.

[14] K. Lau, A. Nordin and K. Ng, "Extracting elements of component-based systems from natural language requirements," in 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011.

[15] F. Nejati, A. A. A. Ghani, N. K. Yap, and A. Jafaar, "PUTRACOM: A Concurrent Component Model With Exogenous Connectors," IEEE Access, vol. 6, pp. 15446–15456, 2018.

[16] F. Nejati, A. A. A. Ghani, N. K. Yap, and A. Jaafar, "Handling state space explosion in verification of component-based systems: A review," arXiv Prepr. arXiv1709.10379, Jul. 2017.