Noname manuscript No.
(will be inserted by the editor)

A Component Model with Verifiable Composition for the Construction of Emergency Management Systems

Tauseef Rana · Muhammad Ali Imran · Abdullah Baz

Received: date / Accepted: date

Abstract Construction of critical systems (e.g. disaster/emergency management systems) demands extra efforts from the developers as compared to non-critical systems. A critical system may be comprised of hardware communication infrastructure and the management system software. Many of hardware failures or in-capabilities can be handled by the resilient software components. The management software's main tasks are to continuously collect data (from distributed sensors), to predict possible threats, and to intimate the right authorities for timely action to avoid/reduce the damages of the threats. For the construction of such systems, by reusing existing reliable software components, a verifiable software composition mechanism is highly desired. In this paper, we select a component model (EX-MAN) from the component-based development approaches for providing pre-defined exogenous connectors. In this paper, at the methodological level, we define an approach to verify the correctness of exogenous connectors. To evaluate our approach, we design an emergency management system in EX-MAN and implement the system in our tool for EX-MAN.

Keywords Disaster Management System · Rapid Development · Verification · Connectors · Sensors · Composition

T. Rana (Corresponding Author)
Department of Computer Software Engineering, MCS, National University of Sciences and Technology (NUST)
Islamabad, Pakistan.
E-mail: tauseefrana@mcs.edu.pk

1 Introduction

Computer-based systems are used in all fields of life and the users of these systems have dependability of different scale on these systems. For high scale dependability, some of the systems are referred to as critical systems. Critical systems are broadly categorized into four different kinds [18]: (i) safety critical, (ii) mission critical, (iii) business critical and (iv) security critical. A critical system is a pair ($\langle H, S \rangle$) of hardware (H) and software (S). In a system, both these elements may be distributed in different locations. The hardware part may be comprised of many different devices/sensors with the control software produced by many different vendors. These days, a system with such heterogeneous devices are referred as internet of things (IOT) [32,5].

For building a system by composing heterogeneous components (hardware and/or software), a verifiable software composition for the construction of such systems is highly desired [49]. Correctness by construction is a system building strategy that goes inline with the verifiable software composition [6,34,11]. In order to check the structural and behavioural correctness of the system architecture, checking the correctness by construction property in a composition mechanism was introduced in [11]. For the construction of such systems, the importance of composition mechanisms become more demanding if these systems are evolving [18]. In this paper, for such critical management systems, our focus is on the verifiable composition mechanisms which can help us creating and maintaining such systems; disaster management system (DMS [16]) and emergency management systems are two examples of these systems.

In the current world economy, devices/sensors in IOT are playing a vital role. Hence, the need for a

system/software development method based on commercial off-the-shelf (COTS) hardware/software components is emerging [5]. From decades, efforts have been made on optimizing the process of software development for such systems. In this quest, a new development paradigm; Component Based Development (CBD) has emerged [50,47]. The rationale behind CBD is to construct large software systems by reusing software components that are already developed and stored in a domain-specific repository of reusable components. Other benefits that can be achieved by reusing already developed software components are; high productivity, low cost, lesser time to market, and quality software. In other words, the essence of CBD is to increase the productivity and quality of the software systems by reducing the time and cost of development [24].

Currently, a number of component models are available [8]; among them, some are based on standardized reuse based software architecture frameworks; such as, Sun's Enterprise Java Beans (EJBs) [9], OMG's CORBA Component Model (CCM) [15], and Microsoft's Component Object Model (COM) [46]. These frameworks articulate the reuse of software artifacts (components/connectors) across development groups. The other class of component models is based on the use of Architectural Description Languages (ADLs). Examples include Fractal, EAST Architecture Language (EAST-ADL), and Architecture Analysis and Design Language (AADL) [17,48,2]. All of these component models cover a wide range of application domains; such as, embedded systems including automotive software systems and consumer electronics, and business domains; such as, finance, telecommunication, healthcare, and transportation [21,7,45,8]. Some other application domains include avionic, reactive systems, safety-critical systems, distributed and parallel systems [17,31]. Each of these domains is highly demanding in a sense that the software system to be constructed by composing already developed software components, using pre-defined composition mechanisms, collectively meets the functional and non-functional requirements of the system [29,21,17,4,22].

In this paper, we select a component model named Extended-X-MAN (EX-MAN) from [39,40]. This model is based on X-MAN component model from [29]. EX-MAN proposes a number of new features to address the limitations of X-MAN. The selected component model uses independent components to be composed by pre-defined connectors for system construction [43]. EX-MAN is a general purpose component model. In this paper, we explore its usefulness for the construction of critical systems. As the verification of partial and final system is an important part for the construction of critical system, we define the proposed methodology for verifying composition in EX-MAN. Furthermore, in this paper, we evaluate our defined methodology by applying it to verify the structural and behavioural correctness of a composite created in EX-MAN. We use EX-MAN tool to build a disaster management system (DMS) and verify our approach by verifying the structural/behavioural correctness of the system. Based on the construction of DMS and its verification by using our approach, we are confident for the usefulness of EX-MAN for constructing critical systems. With this confidence, we intend to invest time and resource to further enhance EX-MAN to address its limitations in the way of constructing critical systems.

The rest of the paper is organized in six sections. Section 2 introduces a disaster management system and describes a case study of this system. Section 3 describes component models for system construction in CBD. The EX-MAN component model with a systematic composition mechanism is described in Section 4. Next, in Section 5, the mechanism of composition verification is defined and described for EX-MAN. A disaster management system from Section 2 is constructed and implemented with a software tool for EX-MAN in Section 6. Finally, the discussion and conclusion is presented in Section 7.

2 Disaster Management System

At a broad level, disasters may be categorized into two types: (i) natural disasters and (ii) man-made disasters. Natural disasters occur due to the natural processes on Earth for no fixed reason or time of occurrence; this category includes heavy rain, earthquakes, lightning, avalanche etc. Man-made disasters occur due to deliberate actions or inabilities to perform certain tasks by human beings. Some of the examples of these actions would be cutting the trees at the banks of a canal which can weaken the canal boundary which can cause floods due to boundary breakage; floods in urban areas can be devastating [19]. For the inability category, a person working on a chemical/atomic plant mistakenly cause a damage in the plant.

A computer based management system to monitor and report any expected causes of a disaster is referred to as the disaster management system (DMS). The purpose of such a system is to convey the information to the right authorities for correct and timely actions to mitigate the damages of any disaster [23]. In this section, we consider a wilderness weather station from [44] example shown in Figure 1.

At the abstract level, the wilderness weather station shown in Figure 1 is comprised of three main subsys-

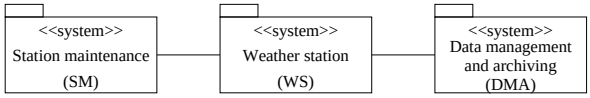


Fig. 1 A wilderness weather station.

tems: (i) Station maintenance (SM), (ii) Weather station (WS), and (iii) Data management and archiving (DMA). There are many weather stations situated at different far locations. These weather stations are maintained remotely. Data is collected from these stations remotely and archived on regular basis.

The context diagram of weather system (to be developed) along with other systems is shown in Figure 2. In this diagram, systems are shown in boxes and the associations between them are shown with lines along with the cardinality information on the associations. This diagram shows that for each weather station (WS) its environment includes a weather information system (WIS), an on-board satellite communication system (SCS) and a control system (CS). The cardinality information shows that there are several weather stations, one control system, one general purpose weather information system and one satellite communication system.

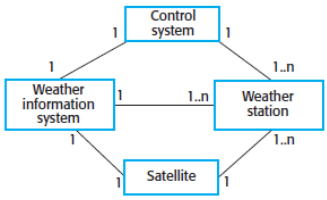


Fig. 2 System context for weather station [44].

The further interaction details with the systems are shown in Figure 3 with the help of a use case model. A use case model represents the interaction with the system; the ellipse shape represents a possible interaction shown in the label in the ellipse shape and the stick figure shows the external entity (a system or a human user) involved in the interaction. The use case represents that WS interacts with WIS for reporting weather data and for reporting WS hardware status. For the use case weather report, the description of the use case is also shown in Figure 3.

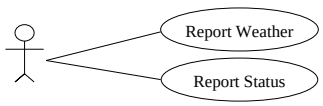


Fig. 3 Use case Diagram–Weather Information System

From the weather station information system use case diagram shown in Figure 3, we consider the report weather use case in this paper. The weather station collects data from different sensors and instruments and sends a summary data to the weather information system on regular basis. In this sent summary, data for minimum, maximum and average values for ground/air temperatures, air pressures, and wind speed are included. Moreover, the total rainfall and wind direction with regular time intervals are sent. This data summary is sent via a satellite communication link with the weather station. Usually, reports of data are requested after every hour but this interval can vary from one station to another.

3 Component Based Development

In CBD, a component model defines an atomic component (a basic building block for reuse [38,1]), composition mechanisms to create composite components (for reuse) and to construct systems (by reuse) for deployment. A component model should support: (i) the development of loosely/uncoupled independent software components, which when used in the construction of a new software system, maximizes the separation of concern [24], (ii) a composition mechanism, that enables the software architects to compose these components hierarchically to achieve compositionality [28] and (iii) ways to construct domain specific software systems for deployment to a certain environment.

In the light of aforementioned, a component model should define the creation of basic/composite components. For a system construction, components are developed using the specific domain knowledge. Currently practiced component models can be categorized into three groups where: (1) components are objects (e.g. EJB [9] and COM [10]) composed by method call (direct message passing), (2) components are UML components [36] or architectural units (AU in architectural description languages (ADLs) [33]) composed by connecting (indirect message passing) matching services/ports, and (3) encapsulated components (e.g. web services [12] and X-MAN components [30,25]) composed by coordination. Considering the DMS from Section 2, we show (in Figure 4) a composite created in different categories of component models. In Figure 4, two components to measure ground temperature (GT) and air temperature (AT) are composed using composition mechanisms from the three categories of component models.

A generic component can be represented by UML notation with provided (represented by a lollipop symbol) and required (represented by a socket symbol) interface of services as shown in Figure 4(b)(i). Objects

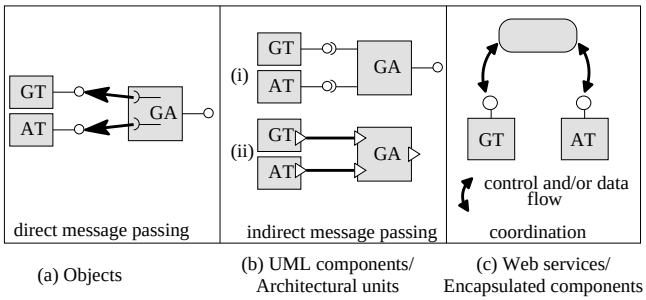


Fig. 4 Composites in CBD.

(with specified provide and unspecified require interfaces) are composed by method call (direct message passing) as shown in Figure 4(a). UML components (with specified provide and specified require interfaces) shown in Figure 4(b)(i) or architectural units (with specified out-port and specified in-port) shown in Figure 4(b)(ii) are composed by connecting (indirect message passing) matching services/ports. Encapsulated components (with specified provide and no require interfaces) are composed by coordination as shown in Figure 4(c). In the DMS composite, for reading values from sensors, these are special components without any required interfaces (unspecified in objects, specified in UML components and specified in-ports).

In Figure 4(a) and Figure 4(b), component GA (for measuring ground and air temperatures)represents a typical component with required interfaces in the respective component model; this component is developed to compose components GT and AT because there is no explicit unit for composing components. Unlike this, the composition in the third category (Figure 4(c)) are composed by a separate program unit to be perform coordination. By the required interfaces, GA component shows a dependency on two components. Objects, UML components and AUs have dependencies on other objects in object based component models (as method calls), on other components in UML and on other AUs in ADLs, respectively. Hence, for composition, using a component of these two categories means availability of all dependent components.

In contrast, encapsulated components have no dependencies. In the third category of component models, there are no pre-defined fixed program units for coordination for the composition of web services. Web services are composed by orchestration [13] to produce a workflow by using BPEL language [35]; orchestration is a form of coordination [37,14], in which participants (web services) are separated from the coordination mechanism. In contrast, X-MAN defines a fixed set of exogenous connectors for adapting and composition of components.

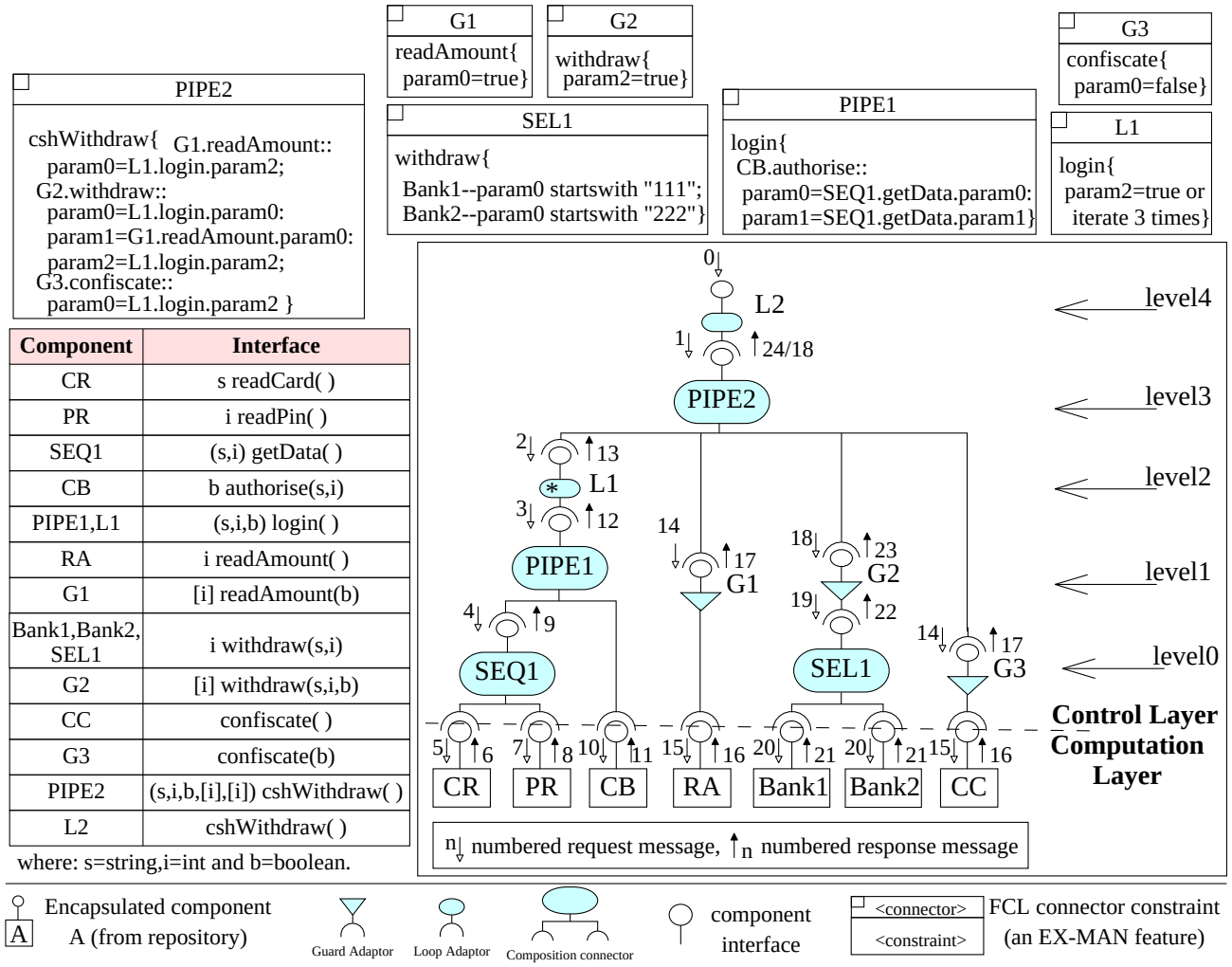
Considering the distinguished feature of encapsulated components and provision of exogenous connectors, we select an enhanced version of X-MAN (EX-MAN) [39,40] which overcomes many of the limitations of X-MAN. In this paper, we define a methodology to verify the correctness by construction with respect to the static structural (interface generation) and the dynamic (control/date flow) behaviour correctness of exogenous connector.

4 EX-MAN Component Model

In this section, we describe EX-MAN model with the help of a Bank system example shown in Figure 5. The reason for the consideration of this example is the usage of all kinds of model elements in one system example. As the theme and objective of this paper is to verify the composition for system construction, we do not cover all details of EX-MAN; details of the model can be found in [39].

The distinguishing feature of X-MAN is the use of exogenous connectors to construct the communication part of a system. In X-MAN, exogenous connectors are defined at an abstract level and the exact behaviours of these connectors are not fixed at the component model level. The semantics of X-MAN exogenous connectors are defined in different tools [26,27,48] in unspecified ways. In order to address this issue, EX-MAN extends the model with constraints written in a simple flow constraint language (FCL) specified for exogenous connectors [42].

A system is a collection of components (shown in the computation layer) and a multi-level hierarchy of exogenous connectors (shown in the control layer). The role of connectors is to pass the request and response in the system. Exogenous connectors are either composition connectors (sequencer represented by SEQ, pipe represented by PIPE and selector represented by SEL) or adaptors (guard represented by G and loop represented by L). A component has an interface that lists the computational services offered by the component. A connector has no interface and in turn no service to offer on its own. However, when connected to one or more component(s), a connector's interface offers either the composite services (services of the composed components; interface of SEQ1) or the adapted service (service of the connected component; interface of G1). Hence, a connector with an interface represents a composite component or an adapted component in the system. Some connectors have constraints as their property. Based on a constraint, a connector has different interfaces and makes different flows for control/data in the system. Creating the interface of a connector is



Component	Interface
CR	s readCard( )
PR	i readPin( )
SEQ1	(s,i) getData( )
CB	b authorise(s,i)
PIPE1,L1	(s,i,b) login( )
RA	i readAmount( )
G1	[i] readAmount(b)
Bank1,Bank2, SEL1	i withdraw(s,i)
G2	[i] withdraw(s,i,b)
CC	confiscate( )
G3	confiscate(b)
PIPE2	(s,i,b,[i],[i]) cshWithdraw( )
L2	cshWithdraw( )

where: s=string,i=int and b=boolean.

Fig. 5 An ATM system in EX-MAN

a static/structural feature and making a specific control/data flow to the connected component(s) is dynamic/behavioural feature of a connector. Components and connectors in EX-MAN are passive elements; an element becomes active to perform its designated role once the control is provided by a service request.

In Figure 5, the system has four composition connectors (sequencer SEQ1, selector SEL1, pipe PIPE1 and pipe PIPE2), five adaptor connectors (finite loop L1, infinite loop L2, guard G1, guard G2 and guard G3) and seven components (CR to read card number, PR to read pin code, CB to authenticate ATM card, RA to read withdraw amount, Bank1, Bank2 and CC to confiscate card). Interface of pre-built components, composites components (created by composition connector) and adapted components (created by adaptors) are shown in a table. FCL constraints of seven connectors are also shown in Figure 5.

Sequencer and pipe connectors passes the control (and data) to each connected components in sequence from left to right. A pipe is a special sequencer that can pass execution result of one component as input data for later component executions. A selector passes control to only one component. A finite loop connector passes control to its connected component for fixed iterations and an infinite loop passes control to its connected component indefinitely. Based on a fixed criteria, a guard connector passes the control to its connected component.

On receipt of request numbered 0 in Figure 5, the first computation of CR component would execute to read the ATM card value and then PR would receive a request to read pin code. The results of these two components are returned as one response numbered 9 by SEQ1 connector. Connector PIPE1 passes the results of these components as inputs to the service request to CB for authentication. The constraint of PIPE1 connec-

tor simply provides mapping between the output values received from the response of first component (composed by SEQ1) to the service of second component CB. Component CB returns boolean value to indicate the authenticity of the inserted card.

Loop L1 is constrained to iterate the service either until the response of CB component is true or to execute to allow user to enter card and pin code three times. The response of L1 is a boolean value to represent the authenticity of the card. PIPE2 is constrained to pass the response received from L1 to services of guards connectors G1, G2 and G3. Based on the authenticity value, these guards are constrained to pass or not to pass control to the connected component. PIPE2 passes control to these guards in sequence. If the authenticity is true then the service of G1 executes to read an amount to withdraw from the user. Next the service of G2 is executed which passes the request to connector SEL1. Based on card data, the request is forwarded to either bank component by this connector. Next, the control is transferred to G3 by PIPE2 and this passes the request ahead if the authenticity value is false to confiscate the card. Lastly, the control is returned to L2 by PIPE2. L2 is an indefinite loop which returns the request back to the first component CR. There is no response out of the system through L2. After serving one customer the ATM system is ready to serve the next customer.

5 Composition Verification

For the system construction in EX-MAN [41], exogenous connectors play a key role; hence, verifying the correctness of these connectors for constructing systems hierarchically is the main objective of this paper. In this section, at the methodological level, we define the exogenous connectors' mechanism to verify the static (for interface generation) and dynamic (for control/data flows) behaviours of exogenous connector. The correctness of the behaviours of connectors would prove the correctness of the system. The hierarchical construction of the control layer of a system (shown in 5 levels in Figure 5) is built recursively and a recursive process is scalable by nature. Hence, the system construction in EX-MAN is scalable; similar is the case with our proposed approach in this paper.

5.1 Conceptual Model of Exogenous Connectors

The focus of EX-MAN is on compositional software design that yields hierarchical (algebraic; composing two components of a type yields another component of the same type [30]) system construction by using exogenous composition connectors. Current component models do not support such kind of system construction [28]. For the system construction, pre-built encapsulated components and exogenous connectors are available in the respective repositories. The correctness of a system is based on the correct composition of components and exogenous connectors to form the operational system. Exogenous connectors play two different core roles in the system construction and in the system operation.

In a system construction, an exogenous connector has the responsibility to create the interface of the composite (e.g. SEQ1 in Figure 5) and adapted (e.g. G1 in Figure 5) components. In the system operation, an exogenous connector has the responsibility to receive a request for the service form its generated interface and to generate requests to the composed and adapted component(s); in other words, the role of the connector is to coordinate the control/data to the connected component(s). At a level of abstraction, to reflect these core responsibilities, we create the conceptual model (Figure 6) for exogenous connectors by using UML class diagram notation.

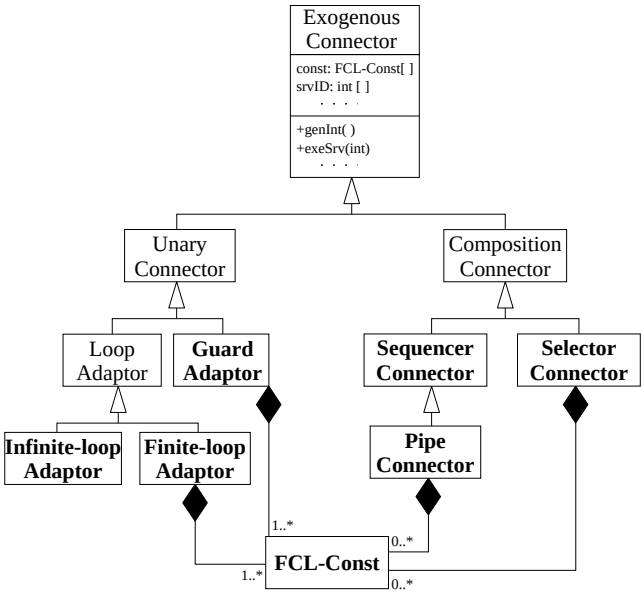


Fig. 6 A conceptual model of exogenous connectors

For the construction of EX-MAN systems, the concrete classes are shown with bold text; rest are the abstract classes. Four connectors can have FCL constraint (instance of FCL-Const) to define the exact behaviour of a connector. A constrained connector is a program unit that translates its specific constraint to define its interface generation behaviour in the system design phase and control/data flows in the system ex-

ecution phase. In EX-MAN, the system construction process is basically based on two steps shown in Figure 7.

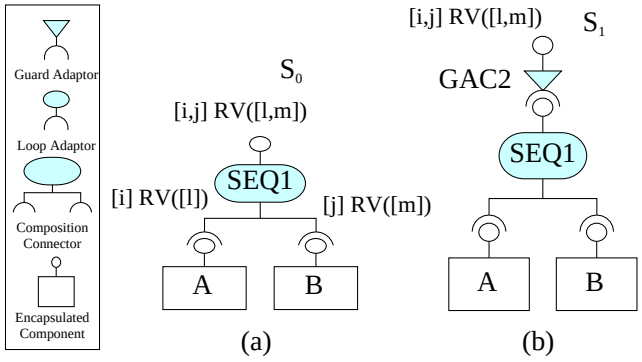


Fig. 7 System construction in EX-MAN.

For system construction, there are two main elements components and exogenous connectors (composition or adaptor) shown in Figure 7. In EX-MAN, a composition connector is n-ary connector by default. An adaptor is either a guard or a loop; these adaptors are represented by different shapes. An adaptor is a unary connector. For the system construction, the two basic steps are to compose and to adapt; in either case, it is a composition of two units.

For the first step in system construction, as a sample, the composition of two components by a sequencer connector is shown in Figure 7(a) to make a partial system S_0; composite SEQ1 in Figure 5 shows a practical use of this composition. For the second step, the out come of the first step (a partial system S_0 shown in Figure 7(a)) is adapted by a guard connector G1 shown in Figure 7(a) to form the next partial system S_1; composite of SEL1 is adapted by a guard in G2 in Figure 5 shows a practical use of this composition. Then thereon, composition of a component or a connector with the partial system and adaptation of the partial system (shown in Figure 7(b)) are possible. For putting two model entities together, in essence, both of these steps are composition. The output of both steps is an EX-MAN partial system which is treated like a component as the partial system can only be connected from the interface of the highest level connector. The composition of a component and the partial system are same in this regards. Hierarchically, an EX-MAN system is an architecture of components (at the lowest level) and many-levels of connectors.

In Figure 7(a), on a system request for service RV with two input arguments, SEQ1 do five actions based coordination in sequence: (i) send a request to A with one input argument 'l', (ii) receive response with one return value 'i', (iii) send a request to B with one input argument 'm', (iv) receive response with one return value 'j' from B, and (v) send response (collections of received responses) for the received request. In Figure 7(b), the partial system of Figure 7(a) is adapted by a guard adaptor with a condition applied on the input argument(s) (e.g. $l > 10$, $m < 5$, $l > 10$ *and* $m < 5$). The invocation of such a partial system is subject to the condition of the guard connector. A loop adaptor can be either terminating or non-terminating. A non-terminating loop connector can only be used at the last (highest) level of connectors; after adapting with the non-terminating loop the construction of the system in ended. In contrast, a terminating loop connector can appear at any level of connectors.

5.2 Static Behaviour of Interface Generation

For the system construction in EX-MAN, in the design phase, a system developer selects components and connectors from the respective repositories. A component has provided services interface and a connector does not as shown in the legend in Figure 5. For composition of any two or more components by a composition connector or for adapting a component by an adaptor, the changed interface of the composite/adapted component is appeared. Based on this interface, the developer can test the partial system and decide to further construct the system. The static behaviour of interface generation by a connector produces this interface.

In the design phase for EX-MAN system construction, an exogenous connector is responsible for generating the interface of the composite/adapted component by the *genInt*() method as shown in the root class in Figure 6. This method defines the static behaviour of the connector. In this section, using Coloured Petri Net (CPN) [20] notations and pseudo code (shown in Figure 8), we define the high level algorithm for interface generation by the exogenous connectors.

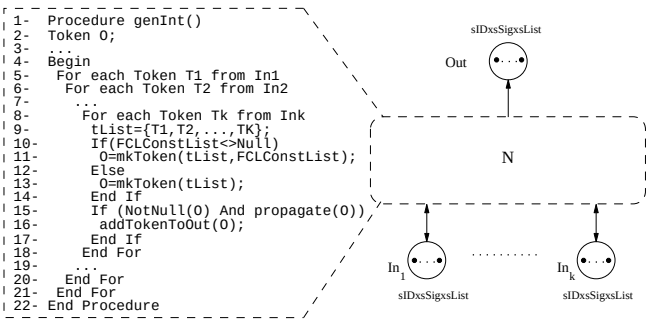
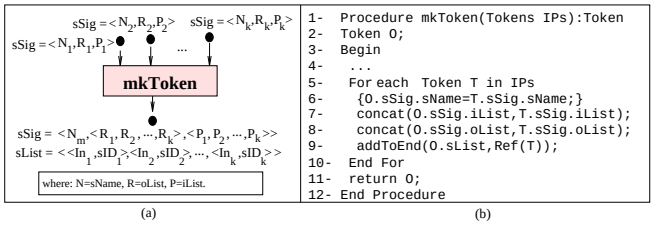


Fig. 8 Generic interface generation mechanism.

In Figure 8, an exogenous connector N may be connected to one (if the connector is an adaptor) or more (if the connector is a composition connector) components; a connected component is represented as an In_i (where i is a numeric value) place of tokens. The interface of the composite/adapted component is represented by the *Out* place with tokens. A token in the shown places is a 3-tuple of a service ID (SID; a numeric value), a service signature ($sSig$ to represent the service name and list of input and output data types) and a list of sub-services ($sList$) referring to services of connected component. The creation of tokens (for *Out* place) is defined by the procedure $genInt()$ with the help of pseudo code. Before adding a created token to *Out* place, the service in the token is checked for a selected service for the composite/adapted component (a special feature of EX-MAN component model [39]).

Figure 8 shows the generic interface mechanism at a level of abstraction for all connectors. However, the mechanism for procedure $mkToken(...)$ is connector specific. As a sample, we consider the $mkToken(...)$ for the sequencer connector as shown in Figure 9. The $mkToken(...)$ procedure is shown schematically in Figure 9(a) and in pseudo code in Figure 9(b). The behaviour of this procedure is to form a combined token from the input tokens.




```
1-  Procedure mkToken(Tokens IPs):Token
2-  Token O;
3-  Begin
4-   ...
5-   For each  Token T in IPs
6-    {O.sSig.sName=T.sSig.sName;}
7-    concat(O.sSig.iList,T.sSig.iList);
8-    concat(O.sSig.oList,T.sSig.oList);
9-    addToEnd(O.sList,Ref(T));
10-  End For
11-  return O;
12- End Procedure
```

(b)

Fig. 9 Sequencer specific operation.

In Figure 9(a), for simplicity, only $sSig$ of input and output tokens, and $sList$ of the output token are shown. In an output token, the service signature contains ordered lists of input/output parameters of the service signatures from the input tokens. The output token contains a list ($sList$) of tuples. The mechanism for naming a compound service is not shown. A compound service is referring to one sub-service of each connected component from left to right.

5.3 Dynamic Behaviour of Operational Coordination

In the execution phase for EX-MAN system construction, an exogenous connector is responsible for generating the request(s) to the composite/adapted component(s) by the $exeSrv(int)$ method as shown in the root class in Figure 6. This method defines the dynamic behaviour of the connector. In this section, using CPN notations and pseudo code (shown in Figure 10), we define the high level algorithm for a service execution by the exogenous connectors.

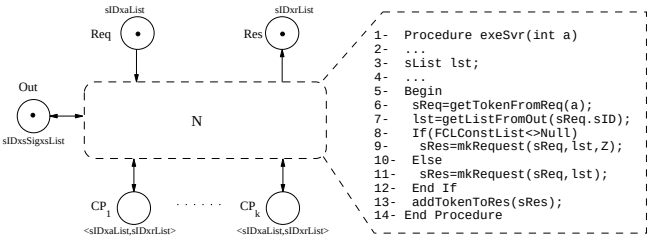


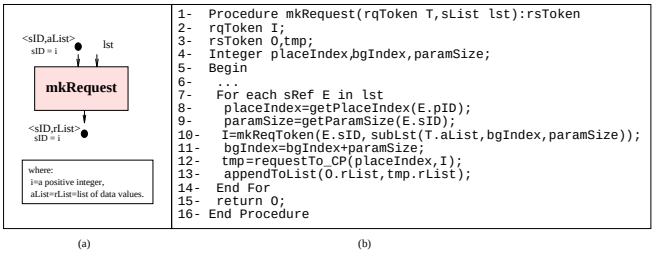
Fig. 10 Generic service execution mechanism.

In Figure 10, an exogenous connector N may be connected to one (if the connector is an adaptor) or more (if the connector is a composition connector) components; a connected component is represented as a CP_i (where i is a numeric value) composition place of tokens. The service request is received as a token in *Req* place and the response is represented by a token in the *Res* place. The place *Out* (from Figure 8) is connected to shown it's use for the request verification.

A token in *Req* place is a token to represent service request with a service ID sID and a list of arguments $aList$ for the requested service. Similarly, a token in *Res* place is a token to represent service response with a service ID sID and a list of response values $rList$ generated by the executed service. The behaviour of a service execution is defined in procedure $mkRequest(...)$.

Figure 10 shows the generic service execution mechanism at a level of abstraction for all connectors. However, the mechanism for procedure $mkRequest(...)$ is connector specific. As a sample, the $mkRequest(...)$ for the sequencer connector is shown in Figure 11. The $mkRequest(...)$ procedure is shown schematically in Figure 11(a) and in pseudo code in Figure 11(b). The behaviour of this procedure is to initiate sub-requests to the referenced sub-services from the connected component(s).

In Figure 11, the argument *lst* to the procedure *mkRequest* of the sequencer connector has two or more tokens. The *mkRequest* procedure finds out the arguments for each sub-service from the *lst* argument, makes request to the sub-service and appends the response of the sub-service into the response token *O*. After getting the response of the last sub-service, the procedure returns the response token to *Res* place.




```
1-  Procedure mkRequest(rqToken T,sList lst):rsToken
2-  rqToken I;
3-  rsToken O,tmp;
4-  Integer placeIndex,bgIndex,paramSize;
5-  Begin
6-   ...
7-   For each sRef E in lst
8-    placeIndex=getPlaceIndex(E.pID);
9-    paramSize=getParamSize(E.sID);
10-   I=mkReqToken(E.sID, subLst(T.aList,bgIndex,paramSize));
11-   bgIndex=bgIndex+paramSize;
12-   tmp=requestTo_CP(placeIndex,I);
13-   appendToList(O.rList,tmp.rList);
14-  End For
15-  return O;
16- End Procedure
```

Fig. 11 Sequencer specific operation.

5.4 Implementation of EX-MAN

We have implemented the EX-MAN component model (described in Section 4) in our prototype tool called Exogenous Composition Framework (ECF). In ECF, using Java language, we have developed two core APIs implementing encapsulated components and exogenous connector (as per the conceptual model shown in Figure 6). The connector API provides classes (shown in Figure 14) for all exogenous connectors. We have verified the semantics of static/dynamic behaviour of all exogenous connectors described in Section 5.2 and in Section 5.3, respectively.

6 Construction of DMS

The system for the interaction between WS and WIS of DMS from Figure 3 is designed using EX-MAN in three construction steps as shown in Figure 12. Components GT (to read ground temperature), AT (to read air temperature), AP (to read air pressure), WiSp (to read wind speed), RF (to read rain fall total value), and WD (to read wind direction value) are composed by using a sequencer connector SEQ1. This composite is referred to as partial system S_0. In the second step, the partial system S_0 is composed with a guard connector G1 to form partial system S_1. In the next step, S_1 is composed with SC (to establish the satellite communication link) by using a pipe connector PIPE1. The output system is referred to as S_2. Instead of using one communication link for data collection using SC1, using a more generic component for communication through many other means is possible.

In Figure 12, for system execution, the flow of requests and responses are shown with numbered arrows. To read the data from all the sensors, a WIS sub-system makes a request to the WS system shown in Figure 12. In the context of the core of this paper, to evaluate the correctness of interface generation and the flow of control/data to the connected component by the exogenous connector, we have designed and executed the system shown in Figure 12 using ECF tool for EX-MAN.

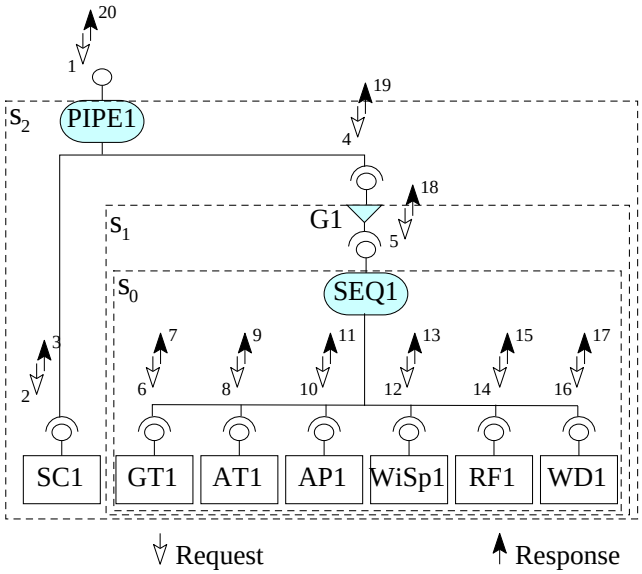


Fig. 12 Weather Station for DMS.

The implementation code of partial system S_1 from Figure 12 is shown in Figure 13. The connector API (used in ECF tool shown in Figure 14) provides classes for all exogenous connectors.

For our future extension of the system, a number of more components and connectors can be used to extend the weather station case study. For frequent data collection from the weather station through the satellite link, a delay component with customizable duration can be composed through a pipe connector and for repetition of the data collection service execution a non-terminating loop connector can be used as the root connector in the system. The component for the delay with customizable duration and the connector for the repetition can be a way to optimize or adapt the system under different situations.

7 Discussion and Conclusion

In EX-MAN, the mechanisms of composition (based on control coordination) are defined by the exogenous composition connectors. In this paper, we have presented a composition verification approach for checking the correctness and EX-MAN exogenous connectors in system construction and in system execution. For constructing critical systems (e.g., emergency management systems), a composition mechanism that can be checked/verified for the correctness of a system construction and execution is a promising and demanded feature.

In the category of component models with exogenous control initiation and management, only EX-MAN defines explicit connectors, which can be used at sys-

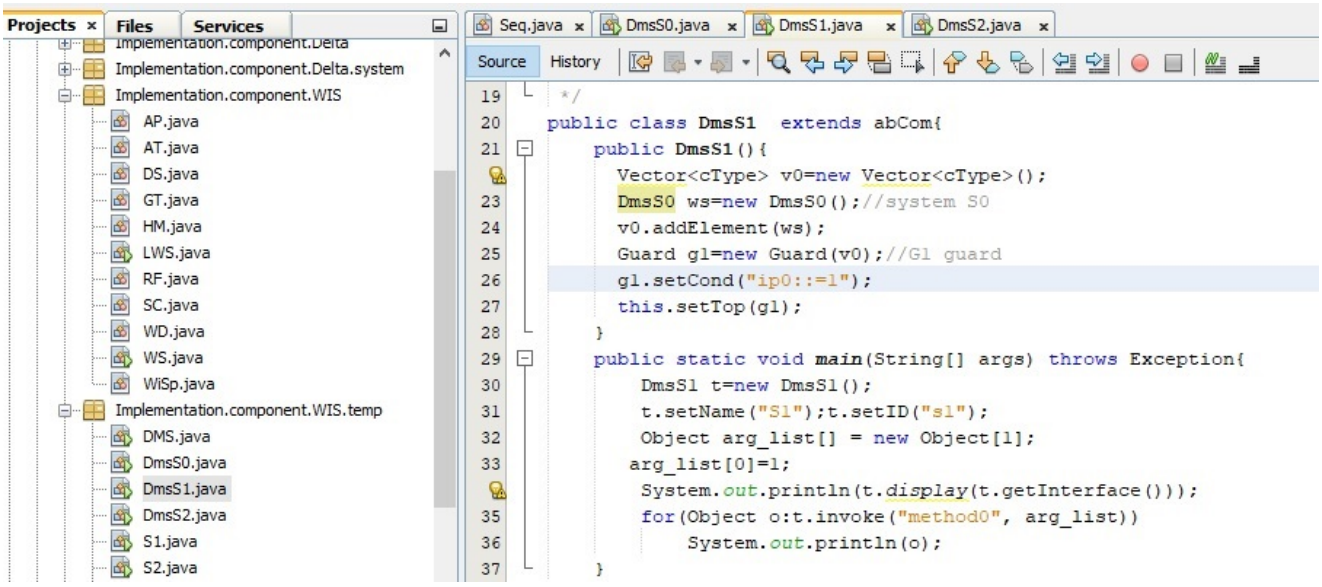


Fig. 13 Execution of S_1 in ECF.

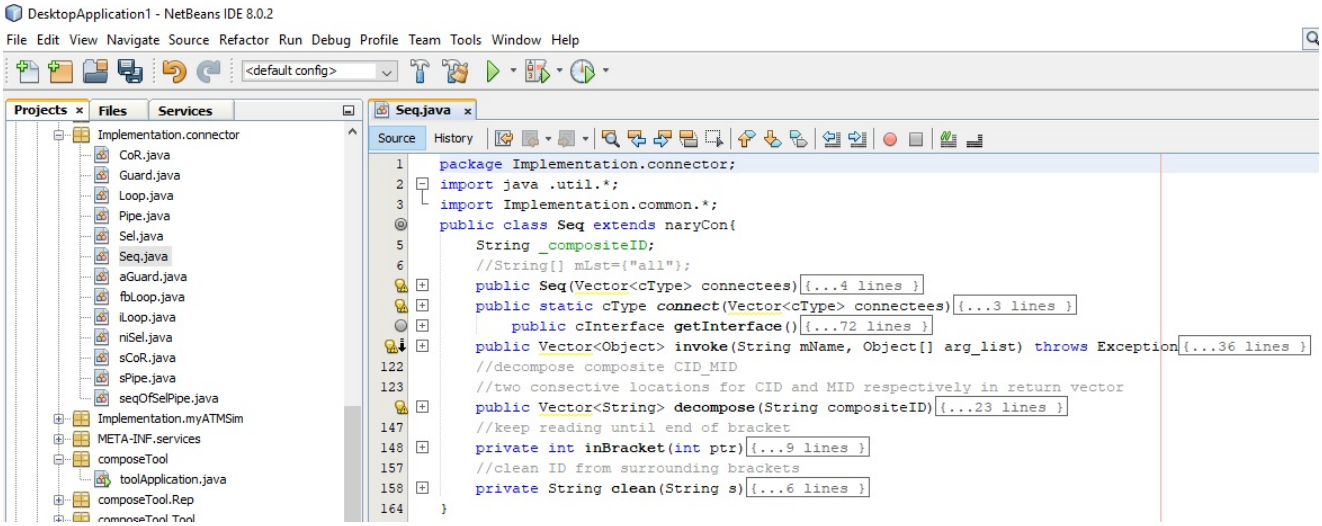


Fig. 14 API for exogenous connectors in ECF.

tem architecture level. These connectors make use of control structures that are Turing complete. Because of this, they help to construct hierarchically (algebraic) complex control coordination structures that are also compositional in nature. Although, some of the component models like Kobra [3] is hierarchical but is not compositional. Current new challenges; i.e., complexity, scalability, and safety faced by CBD demands a system construction that is both hierarchical (algebraic) and compositional in nature. As EX-MAN is specifically designed according to these required lines, it becomes non-trivial to check for the correctness of these exogenous connectors. In this paper, we have defined a mechanism to verify the correctness of these connectors in the system design as well as in system execution.

For the critical management system construction, a component model with verifiable composition is desired. In such a composition, the composite has properties of the composed components. For example, the composition of components (sensors/devices) into a WS composite (shown in Figure 12) makes the composite with the computational properties (to read data from a sensor) of the composed components. Hence, the composite can be checked and verified for the required properties which can be decided when the components are selected for critical system construction. In this paper, using EX-MAN component model, we have showed the construction of DMS in Section 4; exogenous connectors (composition and adopters) are main elements for the system construction. Further in Section 5.2 and Sec-

tion 5.3, we have showed by using a sample exogenous connector that the system construction by exogenous connectors is verifiable and the properties of the system can be predicted from the properties of the components. Hence, based on the DMS construction in Section 6, we conclude that EX-MAN model is suitable for the construction of critical system.

Despite the few strengths of EX-MAN for the construction of critical system, there are a number of limitations associated with this model. Elements (components and connectors) in EX-MAN are passive elements. In EX-MAN system, for a component/connector to perform its tasks the control reaches with a service request. In practice, for critical system, many active components can be used in a system construction. Hence, to address this limitation, we like to extend the model to address this shortcoming in future advancements. To support the development in EX-MAN, we also feel the importance and need of a CASE tool in future advancements.

Acknowledgements This research was partly funded by EPSRC Global Challenges Research Fund–the DARE project–EP/P028764/1. Author Baz would like to thank Umm Al-Qura University (UQU) for the funds he received during projects: 15-COM-3-2-0001, 17-COM-1-01-0009, 18-COM-1-01-0001, 18-COM-1-07-0002, 15-COM-4-1-0001, 15-COM-4-1-0002.

References

1. Achermann, F., Nierstrasz, O.: A calculus for reasoning about software composition. Theoretical Computer Science **331**(2-3), 367–396 (2005)
2. Arbab, F., Baier, C., Rutten, J., Sirjani, M.: Modeling component connectors in reo by constraint automata: (extended abstract). Electronic Notes in Theoretical Computer Science **97**, 25–46 (2004). DOI 10.1016/j.entcs.2004.04.028
3. Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wüst, J., Zettel, J.: Component-based product line engineering with UML. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
4. Belal, M.: Frameworks between components and objects. Advanced Computing: An International Journal **3**, 9–17 (2012). DOI 10.5121/acij.2012.3502
5. Bellekens, X., Atkinson, R., Seeam, A., Tachtatzis, C., Andonovic, I., Nieradzinska, K.: Cyber-physical-security model for safety-critical iot infrastructures. In: Wireless World Research Forum Meeting 35. Copenhagen, Danemark (2016)
6. Chapman, R.: Correctness by construction: A manifesto for high integrity software. In: Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software - Volume 55, SCS '05, pp. 43–46. Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2006). URL http://dl.acm.org/citation.cfm?id=1151816.1151820
7. Cofer, D., Gacek, A., Miller, S., Whalen, M.W., LaValley, B., Sha, L.: Compositional verification of architectural models. In: Proceedings of the 4th International Conference on NASA Formal Methods, NFM'12, pp. 126–140. Springer-Verlag, Berlin, Heidelberg (2012)
8. Crnkovic, I., Sentilles, S., Vulgarakis, A., Chaudron, M.: A classification framework for software component models. IEEE Transactions on Software Engineering **37**(5), 593–615 (2011). DOI 10.1109/TSE.2010.83
9. DeMichiel, L., Yalçinalp, L., Krishnan, S.: Enterprise JavaBeans Specification Version 2.0 (2001)
10. Don, B.: Essential COM, first edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1997)
11. Dong, J., Alencar, P., Cowan, D.: Ensuring structure and behavior correctness in design composition. In: Proceedings Seventh IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS 2000), pp. 279–287 (2000). DOI 10.1109/ECBS.2000.839887
12. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)
13. Fiadeiro, J., Lopes, A., Bocchi, L.: A formal approach to service component architecture. In: Services and Formal Methods, Third International Workshop, WS-FM 2006, pp. 193–213. Springer, Berlin, Heidelberg (2006)
14. Gelernter, D., Carriero, N.: Coordination languages and their significance. Communications of the ACM **35**(2), 96 (1992). DOI http://doi.acm.org/10.1145/129630.376083
15. Group, O.M.: Corba component model 4.0 specification. Specification Version 4.0, Object Management Group (2006). URL http://www.omg.org/docs/formal/06-04-01.pdf
16. Hannan, A., Arshad, S., Azam, M.A., Loo, J., Ahmed, S.H., Majeed, M., Shah, S.: Disaster management system aided by named data network of things: Architecture, design, and analysis. Sensors **18**, 2431 (2018). DOI 10.3390/s18082431
17. He, N., Kroening, D., Wahl, T., Lau, K.K., Taweel, F., Tran, C., Rümmer, P., Sharma, S.: Component-based design and verification in X-MAN. In: Proc. Embedded Real Time Software and Systems (2012)
18. Hinchey, M., Coyle, L.: Evolving critical systems: A research agenda for computer-based systems. In: Proceedings of the 2010 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS '10, pp. 430–435. IEEE Computer Society, Washington, DC, USA (2010). DOI 10.1109/ECBS.2010.56. URL https://doi.org/10.1109/ECBS.2010.56
19. Hsu, S.Y., Chen, T.B., Du, W.C., Wu, J.H., Chen, S.C.: Integrate weather radar and monitoring devices for urban flooding surveillance. Sensors **19**, 825 (2019). DOI 10.3390/s19040825
20. Jensen, K.: Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 2. Springer-Verlag, London, UK (1995)
21. Johnson, K., Calinescu, R., Kikuchi, S.: An incremental verification framework for component-based software systems. In: Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering, CBSE '13, pp. 33–42. ACM, New York, NY, USA (2013). DOI 10.1145/2465449.2465456. URL http://doi.acm.org/10.1145/2465449.2465456
22. Johnson, R.E.: Frameworks = (components + patterns). Commun. ACM **40**(10), 39–42 (1997). DOI 10.1145/262793.262799. URL http://doi.acm.org/10.1145/262793.262799

23. Khaliq, K., Chughtai, O., Shahwani, A., Qayyum, A., Pannek, J.: An emergency response system: Construction, validation, and experiments for disaster management in a vehicular environment. Sensors **19**, 1–23 (2019). DOI 10.3390/s19051150
24. Koziolek, H.: Performance evaluation of component-based software systems: A survey. Perform. Eval. **67**(8), 634–658 (2010). DOI 10.1016/j.peva.2009.07.007. URL http://dx.doi.org/10.1016/j.peva.2009.07.007
25. Lau, K.K., Cola, S.: An Introduction to Component-Based Software Development. World Scientific, Singapore (2017)
26. Lau, K.K., Ling, L., Velasco Elizondo, P., Ukis, V.: Composite connectors for composing software components. In: M. Lumpe, W. Vanderperren (eds.) Proceedings Sixth International Symposium on Software Composition, LNCS 4829, pp. 266–280. Springer-Verlag (2007)
27. Lau, K.K., Ntalamagkas, I., Tran, C., Rana, T.: (Behavioural) Design patterns as composition operators. In: L. Grunske, R. Reussner, F. Plasil (eds.) Proceedings Thirteenth International Symposium on Component-based Software Engineering, LNCS 6092, pp. 232–251. Springer-Verlag (2010)
28. Lau, K.K., Ornaghi, M.: Control encapsulation: A calculus for exogenous composition. In: G. Lewis, I. Poernomo, C. Hofmeister (eds.) Proc. 12th Int. Symp. on Component-based Software Engineering, LNCS 5582, pp. 121–139. Springer-Verlag (2009)
29. Lau, K.K., Ornaghi, M., Wang, Z.: A software component model and its preliminary formalisation. In: F.S. de BoerMarcello M. BonsangueSusanne GrafWillem-Paul de Roever (ed.) Proceedings Fourth International Symposium on Formal Methods for Components and Objects, LNCS 4111, pp. 1–21. Springer-Verlag, Heidelberg Germany (2006)
30. Lau, K.K., Rana, T.: A taxonomy of software composition mechanisms. In: Proceedings Thirty-sixth EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 102–110. IEEE, Lille, France (2010)
31. Lau, K.K., Safie, L., Stepan, P., Tran, C.: A component model that is both control-driven and data-driven. In: Proceedings of the 14th International ACM Sigsoft Symposium on Component Based Software Engineering, CBSE '11, pp. 41–50. ACM, New York, NY, USA (2011). DOI 10.1145/2000229.2000236. URL http://doi.acm.org/10.1145/2000229.2000236
32. Lee, E.: Cyber physical systems: Design challenges. In: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing, ISORC '08, pp. 363–369. IEEE Computer Society, Washington, DC, USA (2008). URL https://doi.org/10.1109/ISORC.2008.25
33. Mehta, N., Medvidovic, N., Phadke, S.: Towards a taxonomy of software connectors. In: ICSE '00: Proceedings of the Twenty-second International Conference on Software Engineering, pp. 178–187. ACM, New York, NY, USA (2000). DOI http://doi.acm.org/10.1145/337180.337201
34. Moriconi, M., Qian, X.: Correctness and composition of software architectures. SIGSOFT Softw. Eng. Notes **19**(5), 164–174 (1994). DOI http://doi.acm.org/10.1145/195274.195403
35. OASIS: Web services business process execution language (2007). URL http://http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
36. OMG: OMG Unified Modeling Language Specification (2007). Http://www.omg.org/cgi-bin/doc?formal/07-11-01.pdf, Access Date: 04-03-2015.
37. Papadopoulos, G., Arbab, F.: Coordination models and languages. Tech. rep., CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands (1998)
38. Proença, J., Clarke, D.: Typed connector families. In: Revised Selected Papers of the 12th International Conference on Formal Aspects of Component Software - Volume 9539, FACS 2015, pp. 294–311. Springer-Verlag, Berlin, Heidelberg (2016)
39. Rana, T.: Incremental construction of component-based systems: A study based on current component model. Ph.D. thesis, School of Computer Science, The University of Manchester (2015)
40. Rana, T.: Ex-man component model for component-based software construction. Arabian Journal for Science and Engineering pp. 1–14 (2019)
41. Rana, T., Bangash, Y., Baz, A., Rana, T., Imran, M.: Incremental composition process for the construction of component-based management systems. Sensors p. 1351 (2020). DOI https://doi.org/10.3390/s20051351
42. Rana, T., Bangash, Y.A., Abbas, H.: Flow constraint language for coordination by exogenous connectors. IEEE Access **7**, 138,341–138,352 (2019). DOI 10.1109/ACCESS.2019.2943164
43. Rana, T., Baz, A.: Incremental construction for scalable component-based systems. Sensors p. 1435 (2020). DOI ttps://doi.org/10.3390/s20051435
44. Sommerville, I.: Software Engineering, tenth edn. Pearson Education Limited, Essex, England (2016)
45. Stepan, P., Lau, K.: Controller patterns for component-based reactive control software systems. In: CBSE'12 - Proceedings of the 15th ACM SIGSOFT Symposium on Component Based Software Engineering—CBSE - Proc. ACM SIGSOFT Symp. Compon. Based Softw. Eng., pp. 71–76. Association for Computing Machinery, United States (2012). DOI 10.1145/2304736.2304749
46. Sullivan, K., Marchukov, M., Socha, J.: Analysis of a conflict between aggregation and interface negotiation in Microsoft's component object model. IEEE Transactions on Software Engineering **25**(4), 584–599 (1999). DOI 10.1109/32.799960. URL http://dx.doi.org/10.1109/32.799960
47. Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming, second edn. Addison-Wesley, New York, NY,United States (2002)
48. Velasco Elizondo, P., Lau, K.K.: A catalogue of component connectors to support development with reuse. The Journal of Systems and Software **83**, 1165–1178 (2010). DOI 10.1016/j.jss.2010.01.008
49. Wang, T.: A context-sensitive service composition framework for dependable service provision in cyber-physical systems. International Journal of Ad Hoc and Ubiquitous Computing **24**, 1 (2017). DOI 10.1504/IJAHUC.2017.10001130
50. Whitehead, K.: Component-Based Development: Principles and Planning for Business Systems. Pearson Education (2002)