A Component Framework where Port Compatibility Implies Weak Termination

Debjyoti Bera, Kees M. van Hee, Michiel van Osch, and Jan Martijn van der Werf

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
{d.bera, k.m.v.hee, m.p.w.j.van.osch, j.m.e.m.v.d.werf}@tue.nl

Abstract. The design and verification of an asynchronous communicating system can be very complex. In this paper we focus on weak termination: in each reachable state, the system has the option to eventually terminate. We present a component framework and construction method that guarantees weak termination. In the framework, communication between components is modeled by portnets, a special class of workflow nets. A basic component defines the orchestration of the portnets. For weak termination, the orchestration should accord to each of the portnets. A composite component is built from basic components that offer some service via a portnet. We provide sufficient conditions to guarantee weak termination for composite components. Furthermore, we present a refinement-based construction procedure to derive a weakly terminating composite from an architectural diagram of the system.

1 Introduction

The class of asynchronous communicating systems encompasses a wide range of software systems that include information systems, embedded systems, grid computing systems, etc. The distributed nature and growing complexity of these systems warrant the need for a component based development (CBD) approach with support for formal analysis techniques. The central idea in the design of such systems involves the construction of complex systems by assembling components while guaranteeing certain properties.

Over the past years, different formal models supporting component based development have been proposed, like Cadena [8] and SaveCCM [5]. Many of these techniques provide a model to specify the components and their composition while relying on state space based explorations to verify the correctness of the design. State space based explorations are generally time consuming and do not scale well to the complexities of real world models. For this reason we construct a framework to guarantee correctness properties by construction. We focus on one property: weak termination.

The weak termination property states that in each reachable state of the system, the system always has the possibility to reach a final state. Generalized soundness [10] is a generalization of weak termination for workflow nets.

A class of generalized sound workflow nets is the class of ST-nets [10] which are constructed by successive refinements of state machines and acyclic marked graphs [6].

Components are loosely coupled. As a consequence, their composition introduces a high degree of concurrency, and thus a state space explosion. In [2] a sufficient condition is presented to pairwise verify weak termination for tree structured compositions. For a subclass of compositions of pairs of components, called ATIS-nets, this condition is implied by their structure [11]. ATIS-nets are constructed from pairs of acyclic marked graphs and isomorphic state machines, and the simultaneous refinement of pairs of places [15].

In this paper, we present a component framework to construct a network of asynchronously communicating components that guarantees weak termination. The framework supports a best practice in communication protocol design: communication between two components is first modeled as a state machine. Then, each transition is assigned to one of the components, such that if any two transitions are in conflict, these transitions are designated to the same component. Then, the state machine is duplicated for each of the components. If a transition is assigned to that component, it sends a message; the corresponding transition of the other component receives this message. Such a net is called a *portnet.* In this way, a component consists of a set of portnets defining its behavior with the environment. A component needs to orchestrate all its portnets, such that for each component it communicates with, it acts as the corresponding portnet. This requirement is similar to the condition imposed by choreography standards like WS-CDL [12]. A Component may be either basic or composite. A basic component provides a service via a portnet. In order to do so, it consumes from other components. In a composition, we allow more than one component to consume a service from another component. Such a composition is a directed graph with edges representing dependency relationships between basic components. If the composition is acyclic, it is a *composite component.*

The orchestration of a component may nest portnets. To resemble this in the architecture, we introduce a simple architectural diagram. Furthermore, we study the behavior of an arbitrary composition of components and give sufficient conditions to guarantee weak termination. We also present a construction procedure based on the rules of [14] and [9] to derive a weakly terminating composition of components.

2 Preliminaries

Let S be a set. We denote the powerset by $\mathcal{P}(S)$. A *bag* over some set S is a function $m : \mathbb{N} \to S$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$ denotes the set of natural numbers. For $s \in S$, $m(s)$ denotes the number of occurrences of s in m. We enumerate bags with square brackets, e.g. the bag $m = [a^2, b^3]$ has an element a occurring twice and element b occurring thrice and all other elements have multiplicity zero. The set of all bags over S is denoted by $B(S)$. We write [] for an empty bag and we use $+$ and $-$ for the sum of two bags and $=, <, >, \leq, \geq$ to element wise compare bags, which are defined in the standard way. A set can be seen as a multiset in which each element of the set occurs exactly once.

A Petri net is a tuple $N = (P, T, F)$, where P is the set of *places*; T is the set of *transitions* such that $P \cap T = \emptyset$ and F is the *flow relation* $F \subseteq (P \times T) \cup (T \times P)$. We refer to elements from $P \cup T$ as *nodes* and elements from F as *arcs*. We denote the places of net N by P_N, transitions as T_N and similarly for other elements of the tuple. If the context is clear, we omit N in the subscript. We define the preset of a node n as $\overset{\bullet}{_N}n = \{m | (m, n) \in F\}$ and the postset as $m^\bullet N = \{n | (m, n) \in F\}$. We lift the notion of a preset and postset to sets: $\overset{\bullet}{_N}S = \cup_{s \in S} \overset{\bullet}{_N}s$ and $S^\bullet_N = \cup_{s \in S} s^\bullet_N$ for some set $S \subseteq (P \cup T)$. If the context is clear, the subscript is omitted. A *path* ν in a Petri net N of length $n \in \mathbb{N}$ is a function $\nu : 1, \ldots, n \rightarrow (P \cup T)$ such that $(\nu(i), \nu(i+1)) \in F$ for all $1 \leq i < n$. We denote a path of length n by $\nu = \langle x_1, \ldots, x_n \rangle$ where $x_i = \nu(i)$ for all $q \leq i \leq n$. The set of all paths of a Petri net N is called the *path space* and denoted by $PS(N)$. Two Petri nets N and M are disjoint if $(P_N \cup T_N) \cap (P_M \cup T_M) = \emptyset$. They are *isomorphic*, denoted by $N \cong_\psi M$ if and only if a bijective function $\psi : P_N \cup T_N \rightarrow P_M \cup T_M$ exists such that $P_M = \psi(P_N)$, $T_M = \psi(T_N)$ and $\forall (x, y) \in F_N \Leftrightarrow (\psi(x), \psi(y)) \in F_M$. We write $N \cong M$ if a bijective function ψ exists such that $N \cong_\psi M$. The state of a Petri net $N = (P, T, F)$ is determined by its marking which represents the distribution of tokens over places of the net. A marking m of a Petri net N is a bag over its places P, i.e., $m \in B(P)$. A transition $t \in T$ is enabled in m if and only if $^\bullet t \leq m$. An enabled transition may fire which results in a new marking $m' = m - {}^\bullet t + t^\bullet$, denoted by $m \xrightarrow{t} m'$. We define the set of reachable markings of a Petri net N with marking m inductively by $\mathcal{R}(m) = \{m\} \cup \bigcup_{m \xrightarrow{t} m'} \mathcal{R}(m')$. We define the *net system* of a Petri net N as a 3-tuple $M = (N, m_0, m_f)$, where $m_0 \in B(P_N)$ is the initial marking and $m_f \in B(P_N)$ is the final marking. The *weak termination* property for a *net system* M states that $\forall m \in \mathcal{R}(N, m_0) : m_f \in \mathcal{R}(N, m)$, i.e. for all reachable markings from the initial marking the final marking is reachable. If a marking does not enable any transition in the net, it is called a *dead marking.* A place is called safe in a net system (N, m_0, m_f) if $\forall m \in \mathcal{R}(N, m_0), m(p) \leq 1$. Let $N = (P, T, F)$ be a Petri net. Net N is a *workflow net* (WFN) if there exists exactly one place $i \in P$ with $^\bullet i = \emptyset$, called the initial place, one place $f \in P$ with $f^\bullet = \emptyset$, called the final place, and all nodes $n \in P \cup T$ are on a path from i to f. The *closure* of a workflow net N is a net $closure(N) = (P, T \cup \{\bar{t}\}, F \cup \{(\bar{t}, i), (f, \bar{t})\})$ such that $\bar{t} \notin T$ and $^\bullet i = f^\bullet = \{\bar{t}\}$. A WFN N weakly terminates if its net system $(N, [i], [f])$ weakly terminates. Note that in [10] this property is called *1-Soundness.* For an overview of soundness, see [1]. Net N is a *state machine* (*S-net*) [6] if and only if $\forall t \in T : |^\bullet t| = |t^\bullet| = 1$. In a state machine, a place p is called a *split* if $p^\bullet > 1$. Likewise, it is a join if $^\bullet p > 1$. Net N is a *marked graph* (*T-net*) [6] if and only if $\forall p \in P : |^\bullet p| = |p^\bullet| = 1$. A workflow net that is also a state machine is called an *S-WFN.* If it is both a workflow net and a marked graph, it is called a *T-WFN.* The class of *ST-nets* were introduced in [10]. These nets allow both concurrency and choice. Note that the class of *T-nets* used in [10] has transitions as the initial and final nodes of the net. We extend such a net to our definition of a *T-WFN* by adding one initial and one final place. The class of *ST-nets* that we will use in this paper includes the class of *S-nets*, *T-nets* and nets obtained after arbitrary successive refinement of places [10] within an *ST-net* by either an *S-WFN* or a *T-WFN*.

3 Component Framework

In this section, we introduce a compositional framework to describe component based systems built of components that are cyclic in their execution and react to inputs from their environment. The main concept of this framework is the notion of a *component*. A component may be *basic* or *composite*. A basic component provides a service and may in turn may use the services offered by other basic components. The interfaces of a basic component are modeled as a *portnet*. A portnet describes a communication protocol. Such a protocol describes all possible sequences of messages that may be exchanged during a service negotiation. A basic component is a closed ST-net providing some service by means of a *sell side portnet* and consuming services using *buy side portnets* from components that have a compatible *sell side portnet*. The sell side portnet of a basic component encapsulates all of its buy side portnets. Furthermore, we allow buy side portnets to be nested. A *composite component* is the composition of a set of pairwise composable basic components such that their dependency graph is *acyclic*.

A *component* is modeled as a Petri net. An *activity* within such a component is modeled by a transition. We distinguish between two types of places, namely *internal places* and *interface places*. An *interface place* is either an *input place* for one component or an *output place* for another component. An *input place* has an empty preset and an *output place* has an empty postset. All other places of a component are referred to as *internal places*. Tokens residing at interface places represent messages, otherwise they are simply *state markers*. Transitions are either *internal transitions* or *interface transitions*. An *internal transition* has no interface places in its preset and postset, whereas an *interface transition* has some interface places in its preset (then it is called a *receive transition*) or its postset (then it is called a *send transition*), but never in both.

3.1 Formalization

Our component framework is based on *open Petri nets* (OPN) which are a subclass of classical Petri nets. OPN are ideal to model communicating systems. This is because they have a distinguished set of *interface places* that represent the interfaces of the net. A direct consequence of the interaction of an OPN with its environment results in tokens being exchanged between these places. Furthermore, we add structural constraints to derive subclasses of an OPN. A *subworkflow net* is a OWN that is a subnet of an OPN.

Definition 1 (Open Petri net, subworkflow net). *An open Petri net (OPN) is defined as $N = (P, I, O, T, F, i, f)$, where* (1) *$P$ is the set of* internal places*;*

(2) I *is the set of* input places *with* ${}^{\bullet}I = \emptyset$*;* (3) O *is the set of* output places *with* $O^{\bullet} = \emptyset$*;* (4) T *is the set of* transitions*;* (5) *the sets* P*,* I*,* O *and* T *are pairwise disjoint;* (6) $i \subseteq P$ *is the set of* initial places*;* (7) $f \subseteq P$ *is the set of* final places*;* (8) $\forall t \in T : {}^{\bullet}t \cap I \neq \emptyset \Rightarrow t^{\bullet} \cap O = \emptyset \wedge t^{\bullet} \cap O \neq \emptyset \Rightarrow {}^{\bullet}t \cap I = \emptyset$*; and* $((P \cup I \cup O, T, F), i,\ f)$ *is the* net system*. We refer to the set* $I \cup O$ *as the* interface places *of the net. The* skeleton *of* N *is a Petri net defined as* $skeleton(N) = (P, T, F')$*, where* $F' = F \cap ((P \times T) \cup (T \times P))$*. The* skeleton system *is defined as* (P, T, F', i, f)*.*

If $skeleton(N)$ *is a* WFN *then* N *is called a* open workflow net (OWN)*. If* $skeleton(N)$ *is a* S-WFN *then* N *is called a* state machine open workflow net (S-OWN)*. If* $skeleton(N)$ *is a* T-WFN *then* N *is called a* marked graph open workflow net (T-OWN)*. If* $skeleton(N)$ *is a* ST-net *then* N *is called a* ST open workflow net (ST-OWN)*.*

Let N *be an OPN and* M *be a OWN. We say that* M *is a subworkflow net of* N *denoted by* $M \sqsubseteq N$ *if and only if* $P_M \subseteq P_N$*,* $T_M \subseteq T_N$*,* $F_M \subseteq F_N$*,* $I_M \subseteq I_N$*,* $O_M \subseteq O_N$*,* ${}^{\bullet}_{N}(T_M \cup O_M \cup P_M \setminus \{i_M\}) \cup (T_M \cup I_M \cup P_M \setminus \{f_M\})^{\bullet}_{N} \subseteq (T_M \cup P_M \cup I_M \cup O_M)$*.*

The transitions of an open Petri net are distinguished into three categories depending on the direction of communication, namely send, receive and internal. A *send transition* contains an output place in its postset. A *receive transition* has an input place in its preset. A transition that does not send or receive is called an *internal transition.*

Definition 2 (Direction of communication). *The* direction of communication *of a transition with respect to a place in an open Petri net* N *is a function* $\lambda : T \to \{send, receive, \tau\}$ *defined as* $\lambda(t) = send \Leftrightarrow t^{\bullet} \cap O \neq \emptyset \wedge {}^{\bullet}t \cap I = \emptyset$*;* $\lambda(t) = receive \Leftrightarrow {}^{\bullet}t \cap I \neq \emptyset \wedge t^{\bullet} \cap O = \emptyset$ *and* $\lambda(t) = \tau$*, otherwise, for all* $t \in T$*. We call a transition* $t \in T$ *a* communicating transition *if and only if* $\lambda(t) \neq \tau$*.*

The refinement of safe places in a Petri net is a well known refinement step and has been described in various contexts [10]. We present here the refinement of a safe place within an OPN by an ST-OWN.

Definition 3 (Place refinement and net reduction). *Given an OPN* N *and an OWN* M *such that* N *and* M *are disjoint, a safe place* $p \in P_N \setminus \{n \mid i_N(n) = f_N(n) = 0\}$ *can be refined by* M*, resulting in an OPN* $N' = N \odot_p M = (P, I, O, T, F, i, f)$ *with* $P = (P_N \setminus \{p\}) \cup P_M$*,* $I = I_N \cup I_M$*,* $O = O_N \cup O_M$*,* $T = T_N \cup T_M$*,* $F = (F_N \setminus (({}^{\bullet}p \times \{p\}) \cup (\{p\} \times p^{\bullet}))) \cup F_M \cup ({}^{\bullet}p \times \{i_M\}) \cup (\{f_M\} \times p^{\bullet})$*,* $i = i_N$*,* $f = f_N$*. We define the reduction of net* N' *by the subworkflow net* M *by* $reduce(N', M) = N$ *if and only if* $N' = N \odot_p M$*.*

An OPN N is said to be *reducible* to another open Petri net N' if and only if successive applications of the reduce operation on net N results in the net N'. Note that we restrict the definition to reductions only by the class of ST-net, since this is the inherent structure of all nets in this component framework. Note that this relation is a preorder.

Definition 4 (Reducible nets). *Consider two OPN's N and N'. We say N is reducible to N' denoted by $N \rightsquigarrow N'$ if and only if $N = N' \vee \exists M : M$ is a ST-OWN $\wedge$ $(M \neq N) \wedge (M \sqsubseteq N) \wedge reduce(N, M) \rightsquigarrow N'$.*

Unlike in an OPN, interfaces in our component framework are more than just a set of interface places acting as message buffers. An interface is determined by a Petri net with a distinguished set of interface places, called the *portnet.* A portnet defines the communication protocol which specifies all acceptable sequences of messages that are permitted to be exchanged over the portnet.

A portnet is an *S-OWN* with structural constraints on the relation between transitions and interface places and paths through it. In a portnet, each interface place is connected to exactly one transition, and each transition is connected to exactly one interface place. Secondly, a portnet must satisfy the *leg property.* A path in a portnet is called a *leg* if it is a path from a split to a join. We also consider the initial place as a split and the final place as a join. The *leg property* requires every leg in a portnet to have at least two transitions with different directions of communication. Lastly, a portnet must satisfy the *choice property*, which requires all transitions belonging to the postset of a place to have the same direction of communication.

Definition 5 (Portnet). *A portnet C is a* S-OWN *such that*

- $\forall t \in T : |({}^\bullet t \cup t^\bullet) \cap (I \cup O)| = 1$;
- $\forall x \in I \cup O : |{}^\bullet x \cup x^\bullet| = 1$;
- (Leg property) $\forall \beta = \langle p_1, t_1 ... t_{n-1}, p_n \rangle \in PS(C) : (|p_1{}^\bullet| > 1 \vee p_1 = i_N) \wedge (|{}^\bullet p_n| > 1 \vee p_n = f_N) : \exists t, t' \in \beta : \lambda(t) \neq \lambda(t')$.
- (Choice property) $\forall t_1, t_2 \in T : {}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset \Rightarrow \lambda(t_1) = \lambda(t_2)$.

We distinguish between two types of portnets: A *sell side portnet* advertises a service and needs a startup message and terminates after sending a result message. A *buy side portnet* consumes a service by sending a startup message and terminates after receiving the result message.

Definition 6 (Portnet types). *Consider a portnet C. We call Portnet C a* sell side portnet *denoted by $sell(C)$ if and only if $\forall t \in i_C{}^\bullet : \lambda(t) = receive \wedge \forall t \in {}^\bullet f_C : \lambda(t) = send$ and we call Portnet C a* buy side portnet *denoted by $buy(C)$ if and only if $\forall t \in i_C{}^\bullet : \lambda(t) = send \wedge \forall t \in {}^\bullet f_C : \lambda(t) = receive$.*

Note that $\neg sell(C) \Leftrightarrow buy(C)$. A *component* is an OPN with a set of portnets. Every communicating transition in a component belongs to a portnet. Furthermore, every portnet of a component is either already a subworkflow net or there exists a subworkflow net that can be reduced to the corresponding portnet.

Definition 7 (Component). *A* component *is a pair (N, Γ) where N is an OPN and Γ is a set of portnets, such that:*

- $\forall t \in T_N : \lambda(t) \neq \tau \Rightarrow \exists C \in \Gamma : t \in T_C$
- $\forall C \in \Gamma : \exists N' : N'$ *is an* $OWN : N' \sqsubseteq N \wedge N' \rightsquigarrow C$

The set of all sell side portnets of a component is defined as: $sellside((N, \Gamma)) = \{C \in \Gamma \mid sell(C)\}$ *and the set of all buy side portnets of a component is defined as:* $buyside((N, \Gamma)) = \{C \in \Gamma \mid buy(C)\}$.

Lemma 8 (Preservation of weak termination). *Consider two OPN's N and M such that $N \rightsquigarrow M$. Then N is weakly terminating if and only if M is weakly terminating.*

Portnets of a component may be nested in each other.

Definition 9 (Nested portnets). *Consider a component (N, Γ) and two portnets $C_1, C_2 \in \Gamma$. We say portnet C_2 is nested in portnet C_1 denoted by $C_2 \lhd_N C_1$ if and only if $\exists M_1, M_2 : M_2 \sqsubseteq M_1 \sqsubseteq N \wedge M_1 \rightsquigarrow C_1 \wedge M_2 \rightsquigarrow C_2$.*

Two portnets are said to be *compatible* if their skeletons are isomorphic. Furthermore, the set of input places of one portnet must match the set of output places of the other portnet while preserving the relation with their associated transitions. Note that a portnet is not compatible with itself.

Definition 10 (Compatible portnets). *Portnets C_1 and C_2 are compatible with respect to some bijective function $\phi : (P_{C_1} \cup T_{C_1} \cup I_{C_1} \cup O_{C_1}) \to (P_{C_2} \cup T_{C_2} \cup I_{C_2} \cup O_{C_2})$, denoted by $C_1 \triangleq_\phi C_2$ if and only if:*

- $skeleton(C_1) \cong_\phi skeleton(C_2)$,
- $O_{C_2} = \phi(I_{C_1})$, $I_{C_2} = \phi(O_{C_1})$,
- $\forall x \in I_{C_1}, t \in T_{C_1} : (x, t) \in F_{C_1} \Leftrightarrow (\phi(t), \phi(x)) \in F_{C_2}$,
- $\forall x \in O_{C_1}, t \in T_{C_1} : (t, x) \in F_{C_1} \Leftrightarrow (\phi(x), \phi(t)) \in F_{C_2}$

We write $C_1 \triangleq C_2$ if a bijective function ϕ exists such that $C_1 \triangleq_\phi C_2$.

Basic components are the building blocks of this component framework. The Petri net structure of a basic component is modeled as an *ST-OWN* with a closure transition. A basic component has one sell side portnet by means of which it provides a service. The sell side portnet may have zero or more nested buy side portnets. Furthermore, each interface place belongs to a unique portnet.

Definition 11 (Basic component). *A* component $B = (N, \Gamma)$ *is a* basic component *if and only if $|i_N| = |f_N| = 1$, N is the closure of an ST-OWN and the following conditions are met:*

- $\forall x \in I_N \cup O_N, \exists! C \in \Gamma : x \in I_C \cup O_C$;
- $\exists C \in \Gamma : i_C = i_N \wedge f_C = f_N \wedge sellside(B) = \{C\}$.

Corollary 12. *Consider a basic component $B = (N, \Gamma)$ and a portnet $C \in \Gamma : sell(C)$. Then $N \rightsquigarrow closure(C)$.*

Note that the closure transition allows the basic component to handle more than one service request. Fig. 1 gives an example of a basic component $M = (N, \Gamma)$, where $\Gamma = \{S1, B1, B2\}$ and $S1$ is a sell side portnet. The sell side portnet has two nested buy side portnets: $B1 \lhd_N S1$ and $B_2 \lhd_N S1$. Net N contains a

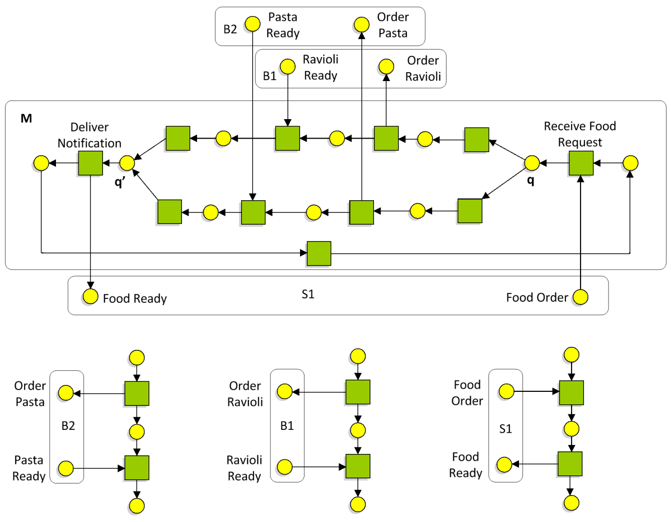

Fig. 1. A basic component

subworkflow net with initial place q and final place q'. This subworkflow net can be reduced by nets $B1$ and $B2$. We refer to the resulting net as an *orchestration net*. Such a net provides the logic behind the order of invocation of the different buy side portnets within a basic component.

Two components are said to be *composable* if and only if the only set of nodes they share are interface places and if this set is not empty, then either they have compatible portnets or they have identical buy side portnets. Note that we require unique sell side portnets.

Definition 13 (Composable components). *Two components $X = (N, \Gamma_N)$ and $Y = (M, \Gamma_M)$ are* composable *denoted by $composable(X, Y)$ if and only if*

- $(P_N \cup I_N \cup O_N \cup T_N) \cap (P_M \cup I_M \cup O_M \cup T_M) = (O_N \cup I_N) \cap (O_M \cup I_M)$;
- $\forall C_1 \in \Gamma_N, C_2 \in \Gamma_M : ((O_{C_2} \cap I_{C_1}) \cup (O_{C_1} \cap I_{C_2}) \neq \emptyset \Rightarrow C_1 \triangleq C_2) \wedge$
  $((I_{C_1} \cap I_{C_2}) \cup (O_{C_1} \cap O_{C_2}) \neq \emptyset \Rightarrow C_1 \cong C_2 \wedge buy(C_1))$.

A composition of a set of pairwise composable components is almost a pairwise union of the tuples of this set, except that the interface places belonging to pairs of compatible portnets, now become the internal places of this composition. Furthermore, the set of portnets of this composition is the set of all incompatible portnets. We extend the composition operation to portnets by treating portnets as components. This is possible in the following way: Consider a portnet C, then this portnet is also a component $(C, \{C\})$.

Definition 14 (Composition of components). *The composition of a set S of pairwise composable components is denoted by $comp(S) = (N, \Gamma)$, where $N = (P_N, I_N, O_N, T_N, F_N, i_N, f_N)$ such that*

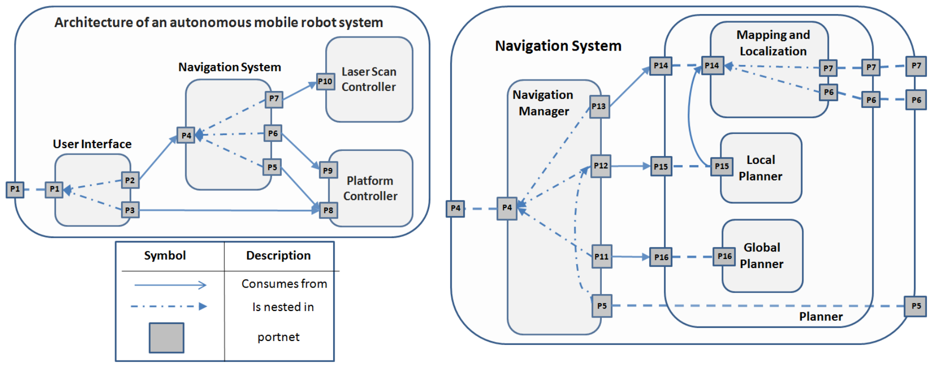

Fig. 2. Architecture diagram

– $P_N = (\bigcup_{(X,\Gamma')\in S} P_X) \cup (\bigcup_{(X,\Gamma')\in S} I_X \cap \bigcup_{(X,\Gamma')\in S} O_X)$,
– $I_N = \bigcup_{(X,\Gamma')\in S} I_X \setminus \bigcup_{(X,\Gamma')\in S} O_X$, $O_N = \bigcup_{(X,\Gamma')\in S} O_X \setminus \bigcup_{(X,\Gamma')\in S} I_X$,
– $T_N = \bigcup_{(X,\Gamma')\in S} T_X$, $F_N = \bigcup_{(X,\Gamma')\in S} F_X$,
– $i_N = \bigcup_{(X,\Gamma')\in S} i_X$, $f_N = \bigcup_{(X,\Gamma')\in S} f_X$, *and*
– $\Gamma = \{C \in \bigcup_{(X,\Gamma')\in S} \Gamma' | \forall C' \in \bigcup_{(X,\Gamma')\in S} \Gamma' : \neg(C \triangleq C')\}$.

Corollary 15. *The composition of a set of pairwise composable components is again a component.*

Note that the composition of one basic component is in fact the basic component itself. Furthermore, $comp(S_1 \cup S_2) \neq comp(S_1 \cup \{comp(S_2)\})$, where S_1 and S_2 are sets of pairwise composable basic components.

3.2 Architectural Diagram

We now present a graphical notation to represent a *composition of components* as an architectural diagram of the system. The diagram abstracts away from the underlying control flow and focuses on the relationships between components and the relationships between the portnets of a component. Components are depicted by a *rounded rectangle*. The portnets of a basic component are represented by a *square*. All entities are labeled. The *dependency relation* between a pair of portnets belonging to different components is represented by a directed arrow indicating the direction of communication initiation, i.e. from a buy side portnet to a sell side portnet. A buy side portnet may have at most one outgoing directed edge while a sell side portnet may have zero or more incoming directed edges. The sell side portnet with zero incoming directed edges becomes the portnet of the composition. The portnets of the composition are represented by extending the portnet with a dotted line to the boundary of the composition. By the structure of a component, we know that all the buy side portnets of a basic component are nested within the sell side portnet. Furthermore, a buy side portnet may nest one or more buy side portnets. We represent the *nesting of portnets* by a dotted directed edge leading from the child to its parent. Note that the Service Component Architecture assembly diagram [3] notation is similar but does not consider nested portnets. We present an architecture diagram of a navigation system in Fig. 2.

4 Behavior

In this section, we study the behavior of a composition of components. In particular, we are interested in weak termination of components, which we define on the skeleton system of the component.

Definition 16 (Weak termination of a component). *A component N is weakly terminating if its skeleton system weakly terminates.*

To prove weak termination for an arbitrary composition of components we first show that the composition of a sell side portnet with a set of compatible buy side portnets is weakly terminating. The crux of the proof relies on both the leg property and the choice property. These properties ensure that every choice and loop is properly communicated to the other compatible portnet, and once the choice to provide a service to a buy side portnet has been made no other buy side portnet can influence the service negotiation. The proof of the following theorem can be found in [4].

Theorem 17. *Let $A, B_1, ..., B_k$ be portnets such that $sell(A)$ and $B_i \triangleq A$ for all $1 \leq i \leq k$, then $comp(\{closure(A), B_1, ..., B_k\})$ weakly terminates.*

Weak termination for an arbitrary composition of portnets is not sufficient to guarantee weak termination for an arbitrary composition of components. To guarantee weak termination for a composition of components, we require the graph of the composition to be acyclic. This is because a cycle indicates a deadlock in the composition. In our framework, we call an acyclic composition of pairwise composable basic components a *composite component*. Note that we will use the shorthand D instead of $D = (N, \varGamma)$, D' instead of $D' = (N', \varGamma')$ and so on, to denote a component without explicitly labeling the tuples. We first introduce the notion of a partner for a buy side portnet in a component, which is the component that provides the compatible sell side portnet.

Definition 18 (Partner). *For a non-empty set S of composable basic components, and the set B consisting of all buy-side portnets of the components of S, we define the function partner : $S \times B \nrightarrow S$ by $\forall D, D' \in S, \forall C \in B : D' = partner(D, C) \Leftrightarrow \exists C' \in \varGamma' : sell(C') \wedge C \triangleq C'$.*

Definition 19 (Acyclic composition, composite component). *Consider a set of S of pairwise composable basic components. Let $R \subseteq S \times S$ be the relation such that $\forall D, D' \in S : (D, D') \in R \Rightarrow \exists C \in \varGamma : D' = partner(D, C)$. The composition is a* composite component *if and only if the transitive closure R^* is irreflexive.*

The result on weakly terminating composition of portnets in conjunction with Lemma 8 allows us to prove that an arbitrary composite component weakly terminates. Both the proof and an example of a deadlock in a cyclic composition can be found in [4].

Theorem 20. *A composite component weakly terminates.*

5 Construction Method

This section presents a construction method that derives a composition of basic components from an architectural diagram and ensures that the derived composition is weakly terminating. The construction method is based on place refinement and composition as defined in the previous sections.

The construction method starts with an architecture diagram of a composite component, like the one depicted in Fig. 2. To construct a basic component we require the three ingredients, namely a sell side portnet, a set of buy side portnets and a set of orchestration nets (ST-WFN). An orchestration net is used to elaborate the activities of a basic component by being able to introduce internal activities, concurrency and choice in a structured way. Furthermore, the places introduced by an orchestration net, may be refined with buy side portnets during construction, thereby allowing us to model both the choice of service invocations and concurrency in service invocations.

First, for each basic component in the diagram, design the sell side portnet. Next, for each basic component in the diagram, derive all its buy side portnets from existing compatible sell side portnets. Note that a buy side portnet may be derived from a sell side portnet by changing the direction of communication associated with each transition in the corresponding sell side portnet. Lastly, for each basic component in the diagram, design the necessary orchestration nets that will be required during the construction.

We may now convert all the sell side portnets into a basic component by introducing the closure transition. For each basic component, the architecture diagram gives the order of nesting of its portnets. Using this information, we may now start designing the control flow of a basic component by successive refinements of an existing internal place with either an orchestration net or a buy side portnet, until all the buy side portnets of the basic component have been added in the right order of nesting and the desired basic component has been constructed.

Construction method

1. Design an architecture diagram for an acyclic composition of basic components using the techniques of Sec. 3.
2. Design all the portnets and orchestration nets that we will need for this composition.
3. For each node in this architecture diagram select the corresponding sell side portnet and apply the closure operation.

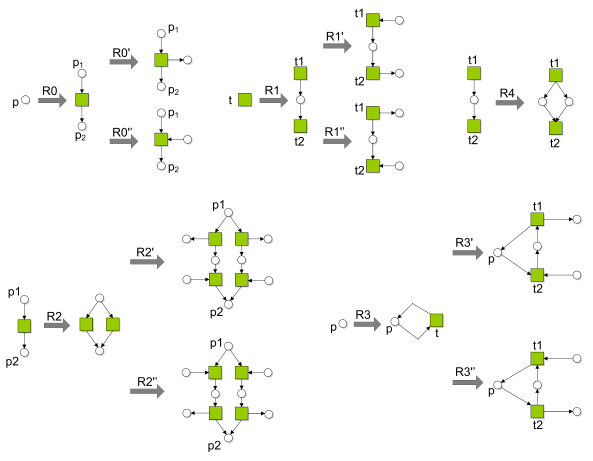

Fig. 3. Refinement rules to generate portnets and orchestration nets

4. For each basic component, repeat the following steps until in each basic component all the buy side portnets have been added in the right order of nesting, and the desired orchestration has been constructed:
   (a) If an orchestration needs to be added first, then choose an internal place and refine with the right orchestration net;
   (b) Otherwise, choose an internal place and refine with a buy side portnet in the order defined by the architectural diagram;
5. Compose the set of basic component using the composition operation.

Theorem 21. *The construction method always results in a composite component that weakly terminates.*

5.1 Construction of Orchestration Nets and Portnets

For the construction of portnets, we extend the Jackson refinement rules $R0$, $R1$, $R2$, and $R3$ with interface places as depicted in Fig. 3. Note that rule $R0$ is a special case of the refinement rule of Def. 3. Rule $R0'$ and $R0''$ extend rule $R0$ such that the refined transition can have either the communication direction send or receive. The extensions of rule $R3$ maintains the leg property by only adding loops with different directions of communication. Similarly, Rule $R2$, which adds a choice to the net is extended such that the choice property is maintained. Rule $R1$ is extended to allow two way communication.

The construction of an orchestration net starts with a single place. By applying the refinement rules of [9], we obtain larger nets that are guaranteed to be weakly terminating. We limit ourselves by applying the Jackson refinement rules $R0, R1, R2, R3, R4$ such that the result remains an ST-net. The construction of a portnet starts with the choice of the *sell side portnet* or *buy side portnet*. A sell side portnet is obtained by the sequence of refinements: $R0; R1; R1'$. A buy side portnet is obtained by the sequence of refinements: $R0; R1; R1''$. We may now further elaborate these portnets by arbitrary applications of the refinement rules $R0; R0'$, $R0; R0''$, $R1; R1'$, $R1; R1''$, $R2; R2'$, $R2; R2''$, $R3; R3'$, $R3; R3''$, while ensuring that the structure of the portnet remains a S-OWN by not allowing for place duplication (rule $R4$). Note that the choice of the place to apply the refinement sequence $R3; R3'$ or $R3; R3''$ must be such that the newly introduced legs do not violate the leg property.

Theorem 22. *The refinement rules for portnets preserve a portnet.*

6 The Control Flow of an Autonomous Mobile Robot

We will model the control flow of the navigation system on a mobile robot. The software system comprises of four main components: The user interface, the navigation system, the platform controller and the laser scan controller. The robot perceives its environment by means of a planar laser scanner. The laser scan controller provides the latest scan as a service. The platform controller is a composite component and provides two services (a) to set a desired velocity (b) queries on the latest odometry. The navigation system is capable of creating a map of its environment and localizing itself on this map using the current laser scan and odometry services. Furthermore, the navigation system can accept a waypoint and generate a sequence of velocity commands that drive the platform to this waypoint while avoiding obstacles. The user interface at the remote location allows an operator to visualize this map and give waypoint to the navigation system. While a waypoint is in progress an operator receives feedback on the progress of this goal. Once the robot has reached its waypoint, the operator is notified. An architecture of the system is presented in Fig. 2.

The navigation system is a composite component comprising of the navigation manager and planning. The latter is again a composite component and comprises of the global planner, the local planner and the mapping and localization system. The mapping and localization system is capable of generating a map and localizing itself using the services offered by the laser scan controller and platform controller. The global planner accepts a map and a waypoint goal and generates a global plan (trajectory) from the robot's current location to the waypoint goal. The local planner accepts a map and a global plan and generates a collision free sequence of velocity commands that drive the robot to the desired waypoint goal. The local planner generates these sequence of velocity commands in a loop until the destination is reached or a valid plan could not be found. In each cycle, the local planner makes use of the mapping and localization system to check its current location and generates feedback on the progress of this goal. If the destination has arrived then this is notified and the planner terminates. If at any moment, a valid velocity command could not be found then this situation is notified and the planner terminates. The navigation manager provides the waypoint navigation as a service to the user interface by orchestrating the components of the navigation system in the right order.

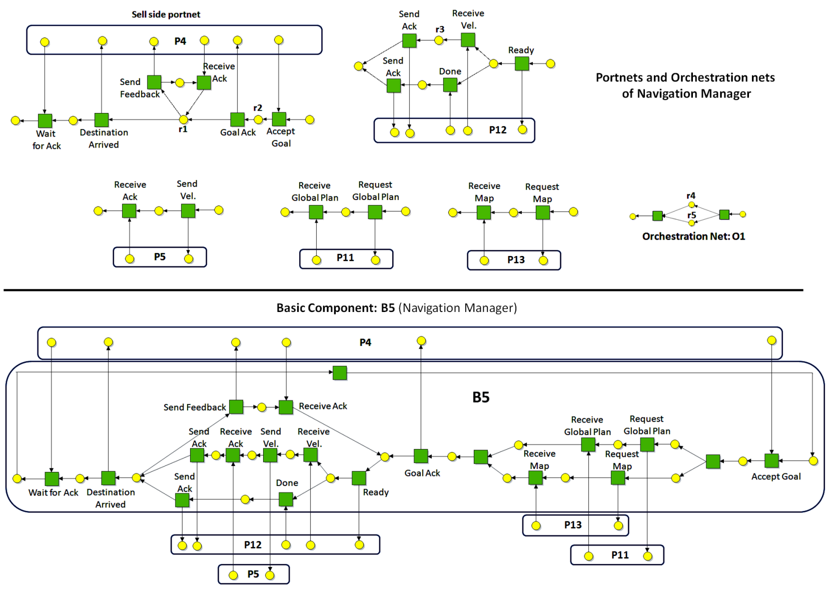

Fig. 4. Basic component: Navigation Manager

The Fig. 4 presents five portents and one orchestration net. From the architecture diagram in Fig. 2, we know portnet $P5$ is nested in $P12$ and all other buy side portnets are nested in the sell side portnet $P4$. We may now apply the construction method to derive the navigation manager in the following way: $(((closure(P4) \odot_{r2} O1) \odot_{r4} P11) \odot_{r5} P13) \odot_{r1} (P12 \odot_{r3} P5)$.

7 Conclusions

In this paper, we introduced a compositional component framework and a construction method to design the control flow of a network of components, while guaranteeing weak termination. The two main concepts of this framework are portnet and basic component. A portnet models the interface of a basic component as a state machine which describes the communication protocol underlying a service negotiation. A basic component provides a service by orchestrating its portnets in the right way. The weak termination property was then investigated by first considering compositions of portnets. It turns out that any pair of compatible portnets that satisfy the *leg property* and the *choice property* always weakly terminate. Furthermore, we prove that an acyclic composition of basic components also known as a composite component weakly terminates.

In [7, 13], the authors focus on constructing deadlock free systems using labeled transition systems, i.e., each component is a state machine, which after composition guarantee deadlock freedom. On the other hand, Petri nets offer a natural way to make formal models of the control flow of a software system. The

Petri net based construction method provides a structured way to design these control flows and guarantee weak termination by construction. In this way they can focus more on the design of each component without having to worry about deadlocks that could be introduced by a composition of components. The designers of software systems can use the guiding principles defined by the construction method during system design .

References

1. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, et al. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.
2. W.M.P. van der Aalst, K.M. van Hee, P. Massuthe, N. Sidorova, and J.M.E.M. van der Werf. Compositional Service Trees. In *ICATPN 2009*, volume 5606 of *LNCS*, pages 283–302. Springer, 2009.
3. M. Beisiegel et al. Service Component Architecture - Assembly Model Specification, SCA Version 1.00, 2007.
4. D. Bera, K.M. van Hee, M.P.W.J. van Osch, and J.M.E.M van der Werf. A Component Framework where Port Compatibility Implies Weak Termination. Technical Report CSR 11-08, Technische Universiteit Eindhoven, 2011.
5. J. Carlson, J. Hakansson, and P. Pettersson. SaveCCM: An Analysable Component Model for Real-Time Systems. *Electronic Notes in Theoretical Computer Science*, 160(1):127 – 140, 2006.
6. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.
7. G. Gossler and J. Sifakis. Component-based construction of deadlock-free systems. In *FSTTCS 2003*, volume 2914 of *LNCS*, pages 420–433. Springer, 2003.
8. J. Hatcliff, W. Deng, M. Dwyer, G. Jung, and V. Prasad. Cadena: An Integrated Development, Analysis, and Verification Environment for Componentbased Systems. In *ICSE 2003*, page 160. IEEE Press, 2003.
9. K.M. van Hee, A.J.H. Hidders, G.J.P.M. Houben, J. Paredaens, and P.A.P. Thiran. On the relationship between workflow models and document types. *Information Systems*, 34(1):178–208, 2009.
10. K.M. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In *ICATPN 2003*, volume 2679 of *LNCS*, pages 337–356. Springer, 2003.
11. K.M. van Hee, N. Sidorova, and J.M.E.M. van der Werf. Construction of asynchronous communicating systems: Weak termination guaranteed! In *Software Composition*, volume 6144 of *LNCS*, pages 106–121. Springer, 2010.
12. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web Services Choreography Description Language Version 1.0. `http://www.w3.org/TR/ws-cdl-10/`, November 2005.
13. K. Klai, S. Tata, and J. Desel. Symbolic Abstraction and Deadlock-Freeness Verification of Inter-enterprise Processes. In *Business Process Management*, volume 5701 of *LNCS*, pages 294–309. Springer, 2009.
14. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
15. J.M.E.M. van der Werf. *Compositional design and verification of component-based information systems.* PhD thesis, Technische Universiteit Eindhoven, 2011.