Construction rules for component-based architectures

Construction Rules for Component-Based Architectures

W.M.P. van der Aalst[1,3], K.M. van Hee[2,3], and R.A. van der Toorn[2,3]

[1] Faculty of Technology Management, Department of Information and Technology, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands. w.m.p.v.d.aalst@tm.tue.nl
[2] Deloitte & Touche Consultants, P.O. Box 23103, NL-1100 DP Amsterdam, The Netherlands. kvanhee@deloitte.nl, rvandertoorn@deloitte.nl
[3] Faculty of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.

Abstract. Software architectures are shifting the focus from lines-of-code towards coarser-grained "business-process-components" and their interconnecting structures. These components are not designed and built from scratch, but composed from smaller components. The challenge in designing these architectures is to preserve construction consistency, meaning that a complete system of components has the same visible behavior as its specifying top-level component. We start with a single component, which is the specification of the system, and we use construction rules to extend the component architecture step by step with new components, while the construction consistency is preserved. In addition we introduce one design pattern. The advantage of using the pattern is that we do not have to verify construction consistency because it is implied by the pattern, i.e., it is introduced as an alternative for state space checking. Finally, we show that if a system is consistent and its specifying top-level component has the option to complete transactions properly, then the complete system has this property as well. The formalisms we use are Petri net theory and branching bisimilarity of transition systems.

Keywords: Software architectures; Component-based design; Correctness-by-construction; Design pattern; Petri nets; Branching bisimulation

1 Introduction

In recent years there has been an increasing interest in *software architectures*, demonstrated by the large number of publications and conferences on this subject and the various commercial activities that are undertaken in this area. This interest can be explained from the development of information systems in the past forty years and the complex status of today's information systems.

In the development of information systems we can identify roughly the following four steps. The first step, starting in the sixties, is that systems are developed as isolated applications dedicated to a single specific purpose. Data is stored in files and the files are only accessed by stand-alone applications. In the second step, starting in the seventies, data is separated from applications, and stored in *Data Base Management Systems*. Shared data stored in databases couples applications. In the eighties, the focus shifts to the end-user of a system. There is an interest for *user interfaces*, visual programming languages and user interface management systems emerge. In the nineties business processes and control flows are separated from other aspects of information systems. This leads to the development of *Workflow Management Systems* [7]. These systems are used to guide users according their business processes along the various transitions they have to fulfill in the system. The result of this development is the separation of concerns and a modular organization of information systems. Moreover these modules are managed by separated control flows.

Today's information systems are often very complex mainly caused by the many technological possibilities to connect systems. *Middleware products* such as *BizTalk*, *BEA Weblogic Integration*, and *Tibco* and *component technology* [44] such as *COM Components*, *CORBA* and *Enterprise Java Beans* facilitate *Enterprise Application Integration* (EAI) by removing technical obstacles. Therefore information systems with different business functionalities and possibility running on different platforms may be connected. For instance a legacy system containing product data can be connected to a Customer Relationship Management application and at the same time disclosed for the internet to facilitate on-line updates. Moreover a Workflow

Management System can be used to coordinate these systems. In fact there are often connections at various levels, which increases the complexity even more: within applications between components, between applications in a single organization, and across company borders. For business-to-business or business-to-government integration *EDI* and *XML* concepts are often used and for business-to-consumer integration the internet plays a major role. Although technology is an enabler for an open systems environment it does not answer all questions. We come across two key questions that remain unanswered: How do we structure a complex network of interconnected systems such that it becomes manageable? And, how do we manage transactions across interconnected systems?

Software architectures are a mean to deal with the complexity of information systems and to reason about various aspects of information systems at various levels of abstraction. In [13] a software architecture is defined "as the structure, which comprises software components, the externally visible properties of those components, and the relationships among them." Software architectures have *functional* and *non-functional* dimensions. The functional dimensions consists of the *structure* and the *behavior* of an architecture. Non-functional dimensions [13] are often captured by the so-called *quality attributes* of an architecture. For instance: performance, security, availability, usability, modifiability, portability, reusability, and integrability. Often these qualities compete and any design decision involves trade-offs. In this paper we consider the functional dimensions of software architectures. In particular we focus on the *dynamic behavior* of components rather than the passing of data, the signature of methods, and naming issues. Dealing with dynamic behavior in a correct and efficient way is an architectural problem. We will use the topology of a *tree* to structure architectures and to express concurrent and interacting components. These components typically have an internal state and interact by sending messages, i.e., asynchronous communication. We reason about correctness of components with respect to their dynamics. The formal basis for modeling and analyzing the dynamics of components is Petri nets [41]. The choice for Petri nets over other formal methods such as process algebra and state charts is primarily motivated by the possibility to describe complex dynamic systems hierarchically [27, 28] and the availability of advanced inheritance notions [4, 5, 14]. Inheritance of behavior is particularly used with respect to refinement and evolution of architectures rather than the reuse of components. A large suite of Petri-net based tools is available to execute Petri-net based models [6]. The foundation of this paper is a particular class of Petri nets called *Component-nets (C-nets)* also referred to as *Workflow-nets* (WF-nets) [1–3]. These nets are very well suited to specify the transactions of a component. Since the framework we introduced in [9] is also based on C-nets, the results of this paper are applicable to this framework. This is advantageous since the dual concept of this framework in which a component has a *specification* and an *architecture* allows for an understanding of the systems functionality by different stakeholders at different levels of abstraction.

There are two different approaches to software design: top down or refinement and bottom up or composition. Given the specification of the component to be built one may refine this component into a network of connected sub-components for which the specifications are derived, or one may look for existing components that can be composed into a component that has a behavior that fits the specification of the original one. In both cases one needs rules that guarantee that connected components preserve some behavioral properties, like the correct treatment of transactions. In this paper we focus on rules to build systems of components. We will use C-nets as elementary building blocks for systems. A system will be a *tree* of C-nets. Figure 1 is an example of a system built from C-nets. In Figure 1 each *rectangle* represents a C-net and each *arrow* represents an *interface* between two C-nets. The C-net A is called the *root* C-net, B, E and F are called the *non-leave* C-nets and C, D, G, H and I are called the *leave* C-nets. In fact one may consider such a tree as one compound component, consisting of the components A, B, C, D, E, F, G, H and I, in which the root component A is the only one that interfaces with the environment. In Figure 1 Y is used to indicate the net which is the composition of B, C and D and Z is used to indicate the net which is the composition of F, G, H and I. The system X itself is a net which is the result of composing Y, E and Z. The root component A is used as the specification for the compound component X. We will prove that under certain conditions a C-net tree (such as X) is a C-net again.

With respect to the behavior of these constructed trees we will require that each non-leave C-net has the same behavior as the composition of this C-net and all its "child" C-nets. In the example of Figure 1 this means that, for instance, B should have the same behavior as the net Y, which is the composition

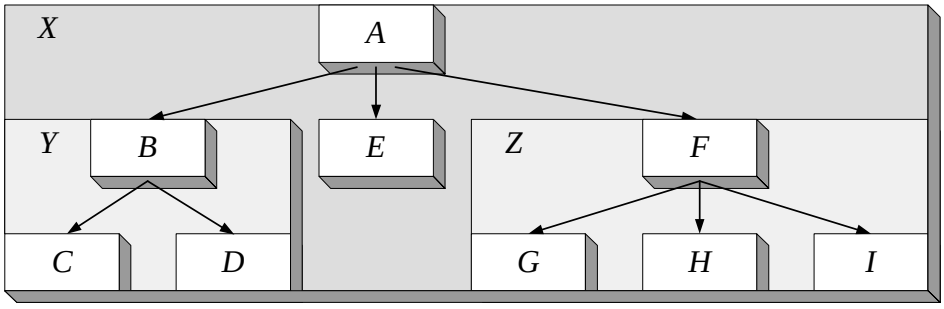

Fig. 1. A C-net tree.

of B and its children C and D. Moreover the behavior of the root component A should be similar to the behavior of X which is the composition of the children Y, E and Z. Therefore applied to the complete C-net tree this requirement means that the observable behavior of the tree is determined by the root C-net. In component-based software development this corresponds to the notion that an interface specification should describe the observable behavior of the complete component. We will prove that if this property holds for all non-leave nodes in the tree, then the tree as a whole behaves as the root C-net. Hence the root C-net is an interface specification for the complete system. We call a tree which has this property *construction consistent* (Definition 24). Construction consistency is an important feature in component-based software development. It ensures that the implementation of a system is consistent with the specification, i.e., respects the users requirements. The main result of this paper is a set of construction rules and a pattern to design complex trees of C-net that are construction consistent.

The *client-server* composition of C-nets is the elementary construction that is key to all construction rules presented in this paper. (In fact a client-server composition is a very simple C-net tree consisting of two nodes: the client-net is the root and the server net is the leave.) To create a client-sever composition we start with a single C-net and we will attach it to another C-net. We will give the example of an embedded software component to inspect and sort out letters. The device has two functionalities: it determines whether a letter is stamped and it scans the address on the letter in order to sort the letter. The component has two subcomponents: one interface component and one technical component. The interface component is described in Figure 2 by *C-net 1* and the technical component is described by *C-net 2*. The composition of the two C-nets is called a client-server composition when these nets are glued together under certain conditions. One of the conditions that is vital to end up with a consistent system is that the behavior of

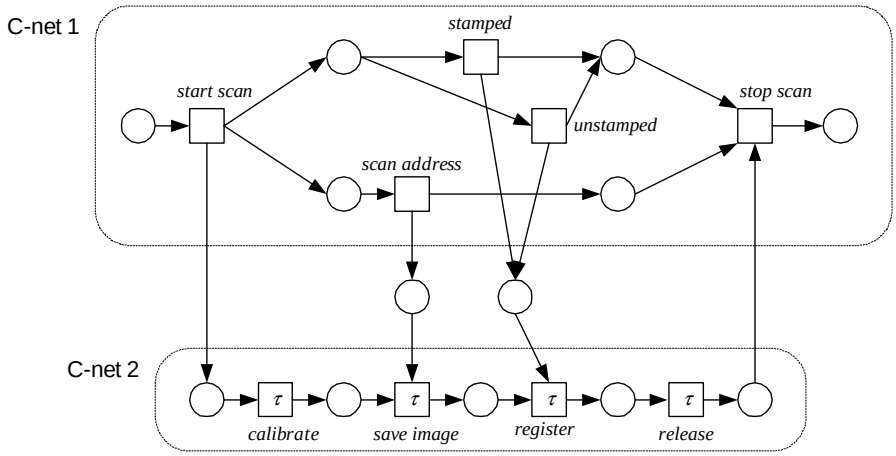

Fig. 2. The interface C-net and the technical C-net glued together.

the client-server composition should be similar to the behavior of the client when we *hide* the behavior of the server; we pretend that *calibrate*, *save image*, *register* and *release* are not visible to the user of the system. In Figure 2 the invisible transitions have been indicated by using the τ-label. When we consider the behavior of the first C-net in isolation, we see that this net has the following functionalities: *start-scan*, *stamped*, *unstamped*, *scan address*, *stop scan*. If the client-server composition of the interface C-net and the technical C-net is able to reproduce exactly the same behavior as the interface C-net in isolation when we hide the behavior of the technical C-net, then we reached our goal, i.e., construction consistency. To compare behaviors of different C-nets we will use the notion of *branching bisimilarity* [25, 36, 38], which is introduced in Section 3.

In addition to the client-server construction we will introduce two other constructions: the *horizontal* and the *vertical union* of client-server compositions. These constructions can be used to extend a C-net tree. Suppose that, for instance in Figure 1, the composition of the nets B and C, respectively B and D are client-server compositions, then the composite net Y consisting of the C-nets B, C and D is called the horizontal union of these two client-server compositions. Moreover, if we suppose that the composition of A and B is a client-server composition, then the composition of A, B and C is called a vertical union.

Finally we present an example of a design pattern which allows building consistent trees: the *request-response pattern*. The advantage of using the request-response pattern is that we do not have to execute an exhaustive state space check on the bisimilarity of the extended tree and its specifying root C-net. Instead we should verify if the conditions of the request-response pattern are satisfied, which are all graph conditions (i.e., no "state explosion" problems). Compare this to the three inheritance-preserving transformation rules based on the notion of branching bisimilarity for sound-processes defined [4, 5, 14]. These rules correspond to design patterns when extending a super class to incorporate new behavior: (1) adding a loop, (2) inserting methods in-between existing methods, and (3) putting new methods in parallel with existing methods. [1]

The structure of this paper is the following. Section 2 presents related work. Section 3 introduces the concepts this paper builds upon. Section 4 discusses the construction rules for the composition of C-nets. Section 5 presents two branching bisimilarity results to obtain construction consistency for C-nets. Section 6 presents a design pattern. Section 7 summarizes the results of this paper.

2 Related work

Software architecture originated in 1968 when Dijkstra [21] pointed out that not only the result but also the structuring of software is important and carries benefits. Parnas contributed by concerning information-hiding modules, software structures [39] and program families. The analogy to a building architecture was due to Alexander [11] who stressed that architectures should be "timeless" and prove to be stable when exposed to the introduction of the latest technologies. Research to software architecture has emerged as a large-scale manifestation over the last years. However the area of software architecture is still quit young considering the large number of definitions that are used simultaneously. Applicable to this paper is the definition by Bass and others [13] quoted in Section 1. This definition is closely related to the de-facto *IEEE 1471* definition [17] agreed upon recently: "A software architecture can be defined as the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution."

Perry and Wolf [40] provided foundation for the study of software architectures and Shaw and Garlan [24, 42] defined and explored the advantages of the use of *architectural style*, i.e., the voluntarily restriction to a relatively small set of choices when it comes to component cooperation and interaction. In this paper, the use of the C-net, the client-server coupling and the request-response pattern are an example of architectural style.

Others stressed the importance of the use of multiple *views*. Views are important because they provide descriptions of the system at various levels of abstraction. Hence they can be used for understanding

[1] For the readers familiar with the work in [9]: A difference with the approach in this paper is that we dropt the a priori requirement of soundness on processes. Instead we handle soundness as a property which can be transferred over branching-bisimilar processes (Lemma 7).

the structure of a system and form a basis for a shared understanding of a system by all its stakeholders. Zachman [45] came up with a view-based approach in which an architecture is considered as a set of architectural representations (or models) placed within two dimensions: the perspective and the description type. The distinguished perspectives agree very well with the interests of different stakeholders in a software development effort. A managers view on the system differs from that of an designer, a programmers view may be completely different from both. Each of the participants views is, however, relevant to develop the system successfully. To capture the various aspects of architectural information Kruchten [32] presented a more concise model of architectural views consisting of the Logical, the *Process*, the *Physical*, the *Development*, and the *Scenarios-view*. If we apply our work to Kruchten, then we can consistently relate the *Conceptual* and the *Process-view* with respect to behavior.

In general architectural problems are addressed by *Architecture Description Languages* (ADLs). For different architectural problems often different ADLs are suited. Medvidovic [35] defined an ADL as a language that provides a concrete syntax and a conceptual framework for characterizing architectures. In [9] we related our approach based on C-nets to other ADLs such as ARMANI, Rapide, Aesop, MetaH, UniCon, Darwin, Wright, C2 and SADL. In contrast with our approach these ADLs typically view software architectures statically [34], i.e., analysis primarily focuses on syntactical and topological issues. Only Darwin offers the possibility to execute "what if" scenarios and Rapide offers a constraint checker based on simulation. Several strategies to compare and to relate ADLs have been presented the last years. One strategy is by using the architecture interchange language ACME [23]. Its purpose is to capture the similarities of ADLs and to support the mapping of architectural specifications from one ADL to another. ACME is suited to do this at the syntactical level, but not at the semantic level. Another strategy to classify ADLs is by *architectural domains*, i.e., the problems or areas of concern that need to be addressed by ADLs [34]. The ADLs investigated in [34] are all supported by tools, which are tightly interwoven with the ADL. Another approach, different to an ADL, but also used to incorporate dynamic behavior is the addition of process specifications to existing middleware technology, e.g., in [16] CORBA IDLs are extended with Petri nets.

The use of patterns to handle architectures problems was first introduced by Gamma [22]. Buschmann [19] also makes use of patterns. An approach by Klein [31] is to reuse reference models.

Notions of behavioral inheritance (also named subtyping or substitutability) are explored by several researchers [12, 29, 30, 33]. Researchers in the domain of formal process models (e.g., Petri nets and process algebras) have tackled similar questions based on the explicit representation of a process by using various notions of (bi)simulation [25, 36, 38].

The theory developed in this paper has applications to *Electronic Commerce* and *Interorganizational Workflow*. References to related work in these areas are given in [5].

3 Preliminaries

3.1 Place/Transition nets

In this section, we define a variant of the classic Petri-net model [20, 37, 41] called labeled Place/Transition nets. The labeled P/T-net is used to describe the *behavior* of a component and its interactions with its environment. Each transition has an action label, either visible or invisible. In the external behavior only the firing of transitions with a visible label can be observed. Let L be a set of visible action labels, and let τ, $\tau \notin L$, be the label used to indicate invisible actions. $L_\tau = L \cup \{\tau\}$. In UML [18] or component-based development [44] visible actions are those actions that are accessible on the interface of a component (i.e. *methods*).

Definition 1 (Labeled P/T-net). *A labeled Place/Transition net is a tuple* (P, T, F, ℓ) *where:*

1. P *is a finite set of* places,
2. T *is a finite set of* transitions *such that* $P \cap T = \emptyset$,
3. $F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs, called the* flow relation, *and*
4. $\ell : T \rightarrow L_\tau$, *is a* labeling function *that assigns a label to a transition,*

such that P, T, F, and L_τ are mutually disjoint.

Let (P, T, F, ℓ) be a labeled P/T-net. Elements of $P \cup T$ are referred to as *nodes*. A node $x \in P \cup T$ is called an *input node* of another node $y \in P \cup T$ if and only if there exists a directed arc from x to y; that is, if and only if xFy. Node x is called an *output node* of y if and only if there exists a directed arc from y to x. If x is a place in P, it is called an input place or an output place; if it is a transition, it is called an input or an output transition. The set of all input nodes of some node x is called the *preset* of x; its set of output nodes is called the *postset*. Two auxiliary functions $\bullet\_, \_\bullet : (P \cup T) \to \mathcal{P}(P \cup T)$[2] are defined that assign to each node its preset and postset, respectively. For any node $x \in P \cup T$, $\bullet x = \{y \mid yFx\}$ and $x\bullet = \{y \mid xFy\}$. The preset and postset functions depend on the context, i.e., the P/T-net the function applies to. If a node is used in several nets, it is not always clear to which P/T-net the preset/postset functions refer. Therefore, we augment the preset and postset notation with the name of the net whenever confusion is possible: ${}^{N}\bullet x$ is the preset of node x in net N and $x{}^{N}\bullet$ is the postset of node x in net N.

We define often classes of objects by a tuple like: "an object y of the class x is denoted by $y = (A, B, C)$". We refer to the elements of the tuple by subscripting them with the name of the object, e.g., A_y, B_y, and C_y. In particular the P/T-net N is denoted by $N = (P, T, F, \ell)$ and the elements are denoted by P_N, T_N, F_N, ℓ_N.

Definition 2 (Labeled P/T-net properties: union, cross-section, sub-net, disjoint).
Let U and W be two labeled P/T-nets such that $(P_U \cup P_W) \cap (T_U \cup T_W) = \emptyset$ and such that, for all $t \in T_U \cap T_W$, $\ell_U(t) = \ell_W(t)$.

1. *The* union *of U and W, denoted $U \cup W$, is the labeled P/T-net $(P_U \cup P_W, T_U \cup T_W, F_U \cup F_W, \ell_U \cup \ell_W)$, where $(\ell_U \cup \ell_W)(t) = \begin{cases} \ell_U(t) & \text{for} \quad t \in T_U \\ \ell_W(t) & \text{for} \quad t \in T_W \backslash T_U. \end{cases}$*
2. *The* cross-section *of U and W, denoted $U \cap W$, is the labeled P/T-net $(P_U \cap P_W, T_U \cap T_W, F_U \cap F_W, \ell_U \cap \ell_W)$, where $\ell_U \cap \ell_W = \ell_U | (P_U \cap P_W)$, i.e., the function ℓ_U restricted to the domain $P_U \cap P_W$.*
3. *U is called a* sub-net *of W, denoted $U \subseteq W$ if $P_U \subseteq P_W$, $T_U \subseteq T_W$ and $F_U \subseteq F_W$.*
4. *If $(P_U \cup T_U) \cap (P_W \cup T_W) = \emptyset$, then U and W are said to be* disjoint.

A *bag* over a set A is a function $A \to \mathbb{N}$. The set of all bags over A is denoted $\mathcal{B}(A)$. Let $X \in \mathcal{B}(A)$, then $X(a)$ denotes the number of occurrences of a in X, often called the cardinality of a in X. The empty bag, which is the function yielding 0 for any element in A, is denoted $\mathbf{0}$. For the explicit enumeration of a bag we use square brackets and superscripts to denote the cardinality of the elements. For example, $[a^2, b, c^3]$ denotes the bag with two elements a, one b, and three elements c. The operations we can perform on a bag are addition, subtraction and comparison. In this paper, we allow the use of sets as bags, in fact we identify in that case a set A with the bag $\{(a, 1) | a \in A\}$, or if $A = \{a_1, \ldots, a_n\}$ A corresponds to the bag $[a_1, \ldots, a_n]$.

Definition 3 (Marking, $\mathcal{N}$). *A* marked, *labeled P/T-net is a pair (N, s), where N is a labeled P/T-net and s is a bag over P denoting the marking (also called state) of the net. The set of all marked, labeled P/T-nets is denoted $\mathcal{N}$.*

Definition 4 (Transition enabling). *Let (N, s) be a marked, labeled P/T-net in $\mathcal{N}$, where N. A transition $t \in T$ is* enabled, *denoted $(N, s)[t\rangle$, if and only if each of its input places p contains a token. That is, $(N, s)[t\rangle \Leftrightarrow \bullet t \leq s$.*

If a transition t is enabled in marking s (notation: $(N, s)[t\rangle$), then t can fire. If, in addition, t has label a (i.e., $a = \ell(t)$ is the associated method, operation, or observable action) and firing t results in marking s', then $(N, s)\ [a\rangle\ (N, s')$ is used to denote the firing.

Definition 5 (Firing rule). *The firing rule $\_\ [\_\rangle\ \_ \subseteq \mathcal{N} \times L \times \mathcal{N}$ is the smallest relation satisfying for any (N, s) in $\mathcal{N}$, with N a labeled P/T-net, and any $t \in T$,*
$(N, s)[t\rangle \Rightarrow (N, s)\ [\ell_N(t)\rangle\ (N, s - \bullet t + t\bullet)$.

[2] $\mathcal{P}(X)$ is the power set of X.

Definition 6 (Firing sequence). *Let (N, s_0) with (N, s_0) be a marked, labeled P/T-net in $\mathcal{N}$. A sequence $\sigma \in T^*$ is called a* firing sequence *of (N, s_0) if and only if $\sigma = \varepsilon$ or, for some positive natural number $n \in \mathbb{N}$, there exist markings $s_1, \ldots, s_n \in \mathcal{B}(P)$ and transitions $t_1, \ldots, t_n \in T$ such that $\sigma = t_1 \ldots t_n$ and, for all i with $0 \leq i < n$, $(N, s_i)[t_{i+1}\rangle$ and $s_{i+1} = s_i - \bullet t_{i+1} + t_{i+1}\bullet$. Sequence σ is said to be* enabled *in marking s_0, denoted $(N, s_0)[\sigma\rangle$. Firing the sequence σ results in the unique marking s, denoted $(N, s_0)\ [\sigma\rangle\ (N, s)$, where $s = s_0$ if $\sigma = \varepsilon$ and $s = s_n$ otherwise.*

Definition 7 (Reachable markings). *The set of* reachable markings *of a marked, labeled P/T-net $(N, s) \in \mathcal{N}$, denoted $[N, s\rangle$, is defined as the set $\{s' \in \mathcal{B}(P) \mid (\exists\, \sigma : \sigma \in T^* : (N, s)\ [\sigma\rangle\ (N, s'))\}$.*

Definition 8 (Directed path). *Let N be a labeled P/T-net. A directed path C from a node n_1 to a node n_k is a sequence $n_1 n_2 \ldots n_k$ such that $(n_i, n_{i+1}) \in F_N$ for $1 \leq i \leq k - 1$.*

Definition 9 (Strong Connectedness). *A labeled P/T-net N is* strongly connected *if and only if, every two nodes x and y in $P_N \cup T_N$ are on a directed path.*

Definition 10 (k-Boundedness). *A marked, labeled P/T-net $(N, s) \in \mathcal{N}$ is* k-bounded *if and only if, for any reachable marking s and any place $p \in P_N$, $s(p) \leq k$ with $k \in \mathbb{N}$.*
A marked, labeled P/T-net $(N, s) \in \mathcal{N}$ is bounded *if and only if the set of reachable markings $[N, s\rangle$ is k-bounded for some $k \in \mathbb{N}$.*

Definition 11 (Dead transition). *Let (N, s) be a marked, labeled P/T-net in $\mathcal{N}$. A transition $t \in T$ is* dead *in (N, s) if and only if there is no reachable marking s' such that $(N, s')[t\rangle$.*

Definition 12 (Deadlock). *Let (N, s) be a marked, labeled P/T-net in $\mathcal{N}$. N is in a* deadlock *in s if and only if there is no $t \in T_N$ such that $(N, s)[t\rangle$.*

Definition 13 (Liveness). *A transition t in a marked, labeled P/T-net $(N, s) \in \mathcal{N}$ is* live *if and only if, for every reachable marking $s' \in [N, s\rangle$, there is a reachable marking s'' such that $(N, s'')[t\rangle$.*
The net is live if all transition in (N, s) are live.

3.2 Component nets

For the modeling of components we use labeled P/T-nets with a specific structure. We will name these nets *component nets* (C-nets).

Definition 14 (C-net, $\mathcal{N}_C$). *A* component net *(C-net) N is a 6-tuple (P, T, F, ℓ, i, o) such that (P, T, F, ℓ) is a labeled P/T-net and the following conditions are satisfied:*

1. *instance creation and completion: P contains two specific places i and o, respectively source and sink place, such that $\bullet i = \emptyset$ and $o\bullet = \emptyset$, and*
2. *connectedness: $\bar{t} \notin (T \cup P)$: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\}, \ell \cup \{(\bar{t}, \tau)\})$ is strongly connected.*

We will denote the class of marked C-nets with $\mathcal{N}_C$.

Analogously to previous notation we denote the source and sink place of a net N with i_N and o_N. The connectedness requirement implies that there is one unique source and one unique sink place. For the readers familiar with the work presented in [2, 15]: C-nets are labeled WF-nets. In this paper we drop the additional requirement we imposed in [9] where start transitions $i\bullet$ and end transitions $\bullet o$ should have a non-τ label. The structure of a C-net allows us to define the following functions.

Definition 15 ($start, stop$). *Let N be a C-net.*

1. *$start(N) = \{t \in T \mid i_N \in \bullet t\}$ is the set of start transitions, and*
2. *$stop(N) = \{t \in T \mid o_N \in t\bullet\}$ is the set of stop transitions.*

Definition 14 only gives a static characterization of a C-net. Components will have a life-cycle which satisfies the following requirements.

Definition 16 (Option to complete). *A marked C-net $(N, [i_N^k])$ has the k-option to complete for some $k \in \mathbb{N}$ if and only if:*

$$\forall s : (s \in [N, [i_N^k]\rangle \Rightarrow [o_N^k] \in [N, s\rangle).$$

If $(N, [i_N^k])$ has the k-option to complete for all $k \in \mathbb{N}$, then N has the option to complete.[3]

This requirement states that starting from the initial marking $[i_N^k]$, i.e., the activation of the component such that there are k tokens placed in the source place i, it is always possible to reach the marking $[o_N^k]$, i.e., the marking with precisely k tokens in place o and all the other places empty, which corresponds to the possibility for a component to terminate without leaving tokens. In principle this leaves the possibility open of the existence of a $s \in [N, [i_N^k]\rangle$ such that $s > [o_N^k]$. The following lemma shows that this can not be the case. Hence the option to complete is in fact the *empty completion option*.

Lemma 1 (No tokens left). *Let N be a C-net. For all $k \in \mathbb{N}$ we have:*
If $(N, [i_N^k])$ satisfies the k-option to complete, then for any reachable marking $s \in [N, [i_N^k]\rangle$ with $s(o) = k$, it follows that $s = [o_N^k]$.

Proof. Suppose that $s(o) = k$ for some $k \in \mathbb{N}$ and let $s \in [N, [i_N^k]\rangle$. Since $(N, [i_N^k])$ satisfies the k-option to complete it follows that $[o_N^k] \in [N, s\rangle$. Suppose there is a place $p \neq o$ with $s(p) > 0$. So there are at least $k + 1$ tokens in the net. C-nets have directed paths to o and hence there is no transition t with $t\bullet = \emptyset$. Hence N keeps at least $k + 1$ tokens and $[o_N^k]$ will never be reached. Contradiction. □

Since Lemma 1 holds we will speak about the *k-empty completion property* instead of the *k-option to complete*, and the *empty completion property* instead of the *option to complete*.

Remark 1 (Relation with soundness notion in [9]). To compare the k-empty completion option used in this paper with the soundness notion in [9], we first extend the k-empty completion option to *k-soundness*.
A C-net N is a *k-sound* for some $k \in \mathbb{N}$ if and only if:

1. $(N, [i_N^k])$ has the *k-empty completion property*,
2. $(N, [i_N^k])$ is *k-bounded*,
3. $(N, [i_N^k])$ contains *no dead transitions*.

Now Lemma 1 implies that 1-soundness, i.e., $k = 1$, corresponds to the soundness notion used in [2, 9, 15].

The motivation for the use of (k-)empty completion property in this paper over the previous 1-empty completion property as a part of the soundness definition in [9] is that it allows for simple analysis for various problems. These problems should have the property that the transitions are equal for all cases. This is typically the case in batch-processing where k cases are handled in parallel as if they were sequentially processed. A necessary condition for these type of processes is that the k-empty completion property is satisfied. Also processes in manufacturing and administrative automation often have this property. For instance consider the example of the assembly process of bicycles. If we have two sets of parts to construct two bicycles, then we may exchange the parts and we still get two bicycles. In a typical assembly process each entry is a single frame of a bicycle and in each step a part is added. The final result at the exit of the process is a complete bicycle. If there are multiple bicycles under construction at the same time, then the k-soundness implies that each entry will yield a bicycle at the end of the process.

An example of an administrative process with multiple tokens is one where the tokens are anonymous. One could think of a process where for instance for ten different couples the same journey is booked. Typical steps are booking flights, hotel rooms, trips, etcetera. In such a process it is not important to which couple which hotel room is attached, as long as each couple will have one in the end.

Another reason for not requiring 1-boundedness in this paper, whereas we did do this in [9], is the following: In [9] we used the 1-boundedness in combination with *activation safeness* to avoid multiple

[3] Note that $[i_N^k]$ and $[o_N^k]$ are *bags* containing k tokens in the input respectively output place of N.

entrance of sub-components, whereas we do allow this in this paper. To overcome the absence of this requirement, we impose the requirement over the *branching bisimulation for C-nets* as explained in the following section.

Not requiring the absence of dead transition is motivated in [10]: In a composition of C-nets it might occur that a dead transition is introduced, whereas they are not present yet in the isolated nets. We can deal with this in two ways. Either we can use a soundness-definition including the absence of dead transitions, or a weaker one without this condition. In the first case we need an operator to remove dead transitions from composite nets. In this paper we choose the second option which is to ignore dead transitions.

3.3 Branching bisimilarity

To compare C-nets we need to formalize a notion of equivalence. In this paper, we use *branching bisimilarity* [25] as the standard equivalence relation on marked, labeled P/T-nets in $\mathcal{N}$.

The notion of a *silent action* is pivotal to the definition of branching bisimilarity. Silent actions are actions (i.e., transition firings) that cannot be observed. Silent actions are denoted with the label τ, i.e., only transitions in a P/T-net with a label different from τ are observable. We assume that τ is an element of L_τ. The τ-labeled transitions are used to distinguish between external, or observable, and internal, or silent, behavior. A single label is sufficient, since all internal actions are equal in the sense that they do not have any visible effects.

To define branching bisimilarity, an auxiliary definition is needed: a relation expressing that a marked, labeled P/T-net can evolve into another marked, labeled P/T-net by executing a sequence of zero or more τ actions.

Definition 17. *The relation* $\_ \Longrightarrow \_ \subseteq \mathcal{N} \times \mathcal{N}$ *is defined as the smallest relation satisfying, for any* $p, p', p'' \in \mathcal{N}$, $p \Longrightarrow p$ *and* $(p \Longrightarrow p' \wedge p' \,[\tau\rangle\, p'') \;\Rightarrow\; p \Longrightarrow p''$.

Let $[(\alpha)\rangle$ be the extension of the relation $[\alpha\rangle$ defined in the following way: for any two marked, labeled P/T-nets $p, p' \in \mathcal{N}$ and action $\alpha \in L_\tau$, $p \,[(\alpha)\rangle\, p'$ if and only if $(\alpha = \tau \wedge p = p') \vee p \,[\alpha\rangle\, p'$.

Thus, $p \,[(\tau)\rangle\, p'$ means that zero τ actions are performed, when the first disjunct of the predicate is satisfied, or that one τ action is performed, when the second disjunct is satisfied. For any observable action $a \in L$, the first disjunct of the predicate is not satisfied. Hence in that case, $p \,[(a)\rangle\, p'$ is simply equal to $p \,[a\rangle\, p'$, meaning that a single a action is performed.

Definition 18 (Branching bisimilarity). *A binary relation* $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ *is called a* branching bisimulation *if and only if, for any* $p, p', q, q' \in \mathcal{N}$ *and* $\alpha \in L_\tau$,

1. $p\mathcal{R}q \wedge p \,[\alpha\rangle\, p' \Rightarrow$
   $(\exists\, q', q'' :\; q', q'' \in \mathcal{N} :\; q \Longrightarrow q'' \wedge q'' \,[(\alpha)\rangle\, q' \wedge p\mathcal{R}q'' \wedge p'\mathcal{R}q')$, *and*
2. $p\mathcal{R}q \wedge q \,[\alpha\rangle\, q' \Rightarrow$
   $(\exists\, p', p'' :\; p', p'' \in \mathcal{N} :\; p \Longrightarrow p'' \wedge p'' \,[(\alpha)\rangle\, p' \wedge p''\mathcal{R}q \wedge p'\mathcal{R}q')$.

Two marked, labeled P/T-nets are called branching bisimilar, *denoted* $p \sim_b q$, *if and only if there exists a branching bisimulation* $\mathcal{R}$ *such that* $p\mathcal{R}q$.

Branching bisimilarity is an equivalence relation on $\mathcal{N}$, i.e., $\sim_b$ is reflexive, symmetric, and transitive. See [14] for more details and references to other notions of branching bisimilarity.
Figure 3 shows the essence of a branching bisimulation. The firing rule is depicted by arrows. The dashed lines represent a branching bisimulation. A marked, labeled P/T-net must be able to simulate any action of an equivalent marked, labeled P/T-net after performing any number of silent actions, except for a silent action which it may or may not simulate.

For marked C-nets we require a stronger concept of branching bisimilarity, namely that the bisimulation relation $\mathcal{R}$ satisfies two additional properties.

Definition 19 (Branching bisimulation for C-nets). *The bisimulation relation* $\mathcal{R} \subseteq \mathcal{N}_C \times \mathcal{N}_C$ *is a* bisimulation relation for C-nets *if and only if* $\forall k \in \mathbb{N}, A, B \in \mathcal{N}_C$*:*

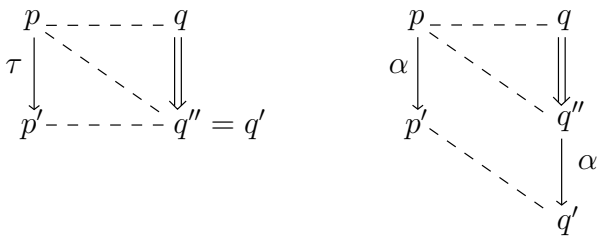

Fig. 3. The essence of a branching bisimulation.

1. existence:
   $(A, [i_A^k])\mathcal{R}(B, [i_B^k])\wedge$
   $(A, [o_A^k])\mathcal{R}(B, [o_B^k])$
2. uniqueness except τ-steps:
   $(\forall x : (A, [i_A^k])\mathcal{R}(B, x) : [i_B^k] \Longrightarrow x)\wedge$
   $(\forall x : (A, [o_A^k])\mathcal{R}(B, x) : x \Longrightarrow [o_B^k])\wedge$
   $(\forall x : (A, x)\mathcal{R}(B, [i_B^k]) : [i_A^k] \Longrightarrow x)\wedge$
   $(\forall x : (A, x)\mathcal{R}(B, [o_B^k]) : x \Longrightarrow [o_A^k])$.

We write $A \simeq_b B$ for the existence of a branching bisimulation relation $\mathcal{R}$ for C-nets. If $A \simeq_b B$, then $(A, [i_A^k]) \sim_b (B, [i_B^k])$ for all $k \in \mathbb{N}$.

This definition allows us to include unreachable states in the relation. However, it is not very useful to consider these states. Therefore we often only have reachable states in mind: $\mathcal{R} \subseteq \{(A, s_A), (B, s_B) | s_A \in [A, [i_A^k]\rangle \wedge s_B \in [B, [i_B^k]\rangle\}$.

The requirements of Definition 19 are automatically satisfied in case both nets have the option to complete. The requirements are a necessity in case one of the processes has not the ability to complete. Figure 4 shows two processes: one without the proper-completion property and one with the proper completion property. In net A in Figure 4(a) one of the transitions x may fire and then a token is either in place p_1 or p_2. The state with one token in the sink place o_A can never be reached. This can also be seen from the state transition diagram by the absence of a transition to the state $[o_A]$. In net B in Figure 4(b) there is only one transition x. When it fires immediately the end state $[o_A]$ is reached. Clearly net B has the proper completion property, whereas net A does not have this property. According to Definition 18 we related the states $[p_1]$ with $[o_B]$, $[p_2]$ with $[o_B]$ and $[o_A]$ with $[o_B]$. It is allowed to relate $[o_A]$ with $[o_B]$ because they are both deadlocks. The process A and B are branching bisimilar with respect to Definition 18, however with respect to Definition 19 they are not, because system A can deadlock in a state which is not the final one, while system B can only deadlock in the final state.
If we would have the proper completion property in both nets, we should have related $[o_A]$ and $[o_B]$. Clearly this relation is not possible in this example. However when we *require* the existence of this relation we are able to show that the proper completion property transfers over branching bisimilar nets (cf. Lemma 7).

3.4 Abstraction and projection inheritance

To compare labeled P/T-nets we use the mechanism of *abstraction*. Abstraction means that we hide the effects of certain action labels. For instance, when we compare two marked labeled P/T-nets x and y, we might abstract from the visible labels in y that are not present in x. We used abstraction in the example of Figure 2 to hide the functionality of the technical component. In [9] the mechanism of abstraction is used to define *projection inheritance* [14]. The idea of projection inheritance can be characterized as follows: "If it is not possible to distinguish the behaviors of two marked labeled P/T-nets x and y, when only transitions of x that are also present in y are executed, then x is a sub class of y."
Abstraction is defined by using an abstraction operator which redefines the labeling function of a P/T-net such that a visible label is replaced by an invisible label. In this paper we will only apply the abstraction

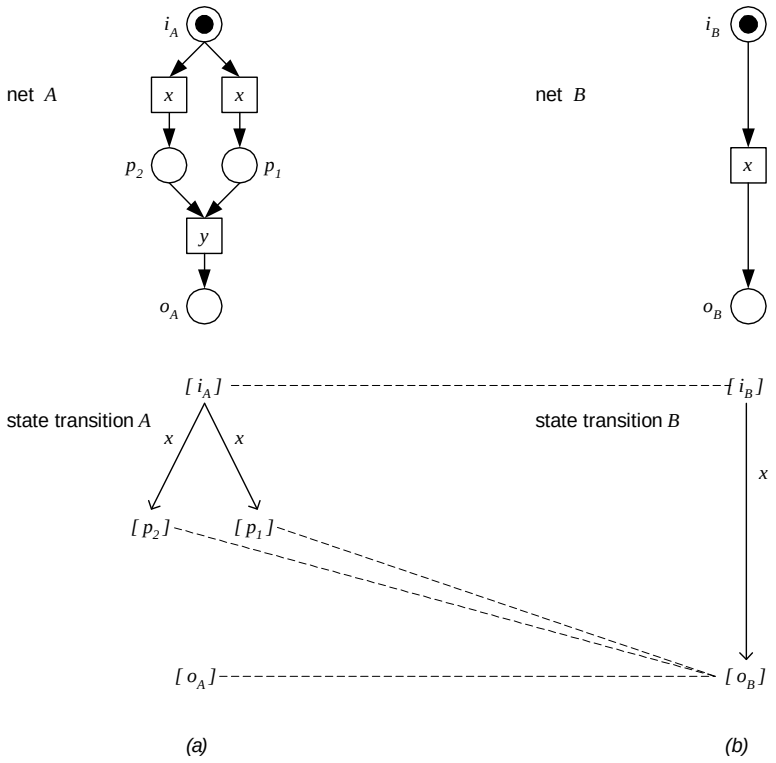

Fig. 4. Two related processes, one with and one without the empty completion property. Both nets and both state transition diagrams are depicted.

operator on entire P/T-nets. We will introduce this operator in the following definition and since it relabels all transitions to the τ-label we will use the same τ-symbol for the operator.

Definition 20 (Abstraction operator). *Let N be a labeled P/T-net. The* abstraction operator τ *is a function on labeled P/T-nets such that $\tau(N) = (P, T, F, \ell')$ where $dom(\ell) = dom(\ell')$ and $\ell'(t) = \tau$ for all $t \in T$, i.e., τ renames all transition labels in N to the silent action τ.*

4 Construction rules for C-nets

In this section we consider compositions of C-nets. In our approach to software design we consider components as independent parts of an architecture, each with their own thread of control and collaborating (by message exchange) to form a working system. For the corresponding C-nets this implies that we do not want to extend their behavior when we put them in an environment. Therefore, when we connect C-nets, we use an interface that does not *extend* the behavior of the separate nets. In fact in Section 5 we give conditions such that we do not *limit* the behavior of the client-nets as well. The interface is introduced in this section. Another design approach we explore in this paper is the *client-server* approach. This implies that in the relationship between connected components there is always one component which has the role of *control component* and one component that has the role of *server component*, i.e., it is obliged to execute tasks for the control component. This client-server approach is reflected in the additional requirements we put on an interface.

We start by introducing an elementary client-server composition and then we define unions of elementary compositions. The unions enable us to create complex trees of connected C-nets. We end this section by introducing a complete tree of C-nets; a structure that can be used to describe a complete component-based

system. The elementary composition is a composition of two C-nets and is called a *client-server* composition because one C-net, the client, governs the transaction, and the other C-net, the server, delivers a service to its client. The transaction that corresponds to this service is always executed within the scope of the transaction of the client. Only the client has the ability to start its server and the server always reports back to the client when it finished its transaction. Figure 5 illustrates the elementary client-server composition.

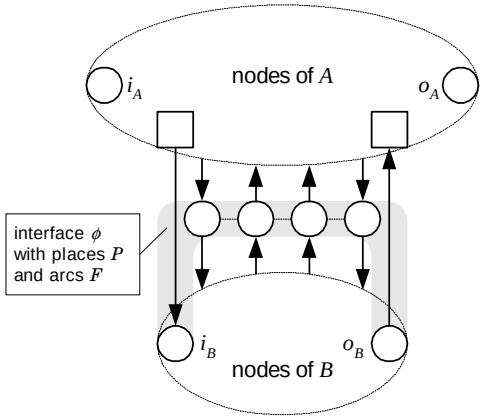

Fig. 5. Client-server composition of C-nets.

We require that this composition is a C-net. The figure depicts the two separate C-nets, A and B. The inner structures of nodes of A and B are left out. Places that are connected to transitions in A *and* transitions in B are called *interface places* and in the figure these places are inside the shaded area, and denoted with P. Together with a set of *interface arcs* F these places are the glue between the nets A and B and are the two components of an *interface* ϕ. An additional requirement we may put on interface is that the start and stop places i_B and o_B of B are also interface places. In that case B is activated by a transition in A and after deactivation of B A may continue. Hence by construction A controls the start and stop behavior of B. The net in Figure 5 is the result of the composition of two nets and one interface.

Definition 21 (Interface, Client-server coupling, Composition operator). *Let A and B be labeled P/T-nets. Let $\phi = (P, F)$ with $F \subseteq (P \times (T_A \cup T_B)) \cup ((T_A \cup T_B) \times P)$ and let $N = (P_A \cup P_B \cup P, T_A \cup T_B, F_A \cup F_B \cup F, \ell_A \cup \ell_B)$.*

1. *ϕ is an* interface *of A and B if and only if $\forall p \in P_\phi$:*
   (a) $\overset{A}{\bullet}p = \emptyset \Leftrightarrow \overset{B}{\bullet}p \neq \emptyset$,
   (b) $\overset{A}{\bullet}p = \emptyset \Leftrightarrow p\overset{A}{\bullet} \neq \emptyset$,
   (c) $\overset{B}{\bullet}p = \emptyset \Leftrightarrow p\overset{B}{\bullet} \neq \emptyset$.
2. *N is a* client-server composition *of A, ϕ and B if ϕ is an interface.*
3. *The function $*$ with $*(A, \phi, B) = N$ is called the* composition operator *and N is denoted by $A *_\phi B$.*

Clearly $A *_\phi B$ is a labeled P/T-net. By definition ϕ is a *place* interface which means that only *transitions* in A and B are externally connected. The requirements on ϕ imply that there is no possibility to "pass-back" tokens to the same net that produced them. Since there are no arcs from transitions in T_B to places in P_A and not from transitions in T_A to places in P_B the connections only *limit* the behavior of the A-part respectively the B-part in $A *_\phi B$. Before we continue we make the following assumption.

Assumption *In the remainder of this paper we assume that there are no name clashes, i.e., all C-nets and interfaces use different identifiers for places and transitions.*

Definition 22 (Union of Interfaces). *Let ϕ and φ be two interfaces with $P_\phi \cap P_\varphi = \emptyset$ and $F_\phi \cap F_\varphi = \emptyset$, then the* union of the interfaces*, denoted $\phi \cup \varphi$, is defined by $(P_\phi \cup P_\phi, F_\phi \cup F_\varphi)$.*

The union of an interface is again an interface. Without proof we state the following lemma.

Lemma 2 (Properties of the client-server composition). *Let A, B and C be labeled P/T-nets with interfaces ϕ between A and B, φ between A and C, and ψ between B and C.*

1. $(A *_\phi B) \cup (A *_\varphi C) = A *_{(\phi \cup \varphi)} (B \cup C)$,
2. $(A *_\phi B) \cup (B *_\psi C) = A *_\phi (B *_\psi C) = (A *_\phi B) *_\psi C$*, i.e., the operator is associative.*

$A *_{(\phi \cup \varphi)} (B \cup C)$ *is called the* horizontal union *and $A *_\phi (B *_\psi C)$ is called the* vertical union.

Note that in general $A *_\varphi B \neq B *_\varphi A$, i.e., the operator is not symmetric.

Lemma 3 (Composition of C-nets is a C-net). *Let A and B be C-nets and ϕ an interface of A and B and let $(P_A \cup P_B) \cap P_\phi = \{i_B, o_B\}$. Then $A *_\phi B$ is a C-net.*

Proof. We verify requirement (1) and (2) stated in Definition 14.

1. By second requirement of Definition 21 it follows that all places in P_ϕ have at least one input and output arc in $A *_\phi B$ to transitions in A and B, hence we only need to consider the input and output arcs of the places in $P_A \cup P_B$. Since A and B are C-nets the only places that do not have both an input and an output arc in respectively A and B, are the places i_A and o_A and i_B and o_B. Composing A and B does not remove any arcs in A or B, therefore in $A *_\phi B$ also i_A, o_A, i_B and o_B are the only possible source and sink places.
   By Definition 21 i_B and o_B are also places of the interface and therefore in $A *_\phi B$ they both have input and output arcs. Only i_A and o_A remain as the source and sink place.
2. It is sufficient to show that for any node x in $A *_\phi B$ there is a directed path from i_A to x and x to o_A. When x is a node of A, then this is true since A is a C-net. There is a directed path from i_A to i_B because i_B is output place of some transition in A and there is a directed path from o_B to o_A because o_B is input place of some transition in A. Therefore, when x is a node of B, it is true because B is a C-net. Finally we have to show that all places in P_ϕ are on a directed path from i_A to o_A. This follows from the the fact that all input and output transitions of $p \in P_\phi$ have this property. □

Next we will consider unions of client-server compositions. We will extend the client-server composition in the "horizontal" and "vertical" direction. The combination of both unions is a handle to build trees of C-nets.

Lemma 4 (Horizontal union of client-server compositions is a C-net). *Suppose that $A *_\varphi B$ and $A *_\psi C$ are client-server compositions of the C-nets A, B and C with the interfaces φ and ψ and let $(P_A \cup P_B) \cap P_\varphi = \{i_B, o_B\}$ and $(P_A \cup P_C) \cap P_\psi = \{i_C, o_C\}$, then $A *_{(\varphi \cup \psi)} (B \cup C)$ is a C-net called the* horizontal union*.*

Proof. We have: $A *_\varphi B \cup A *_\psi C = (P_A \cup P_B \cup P_C \cup P_\varphi \cup P_\psi, T_A \cup T_B \cup T_C, F_A \cup F_B \cup F_C \cup F_\varphi \cup F_\psi, \ell_A \cup \ell_B \cup \ell_C)$. By the assumption that there are no name clashes it follows that this labeled P/T-net is well defined. We now prove that $A *_{(\varphi \cup \psi)} (B \cup C)$ is a C-net.

1. $A *_\varphi B$ is a C-net with source place i_A and sink place o_A and $A *_\psi C$ is also a C-net with source place i_A and sink place o_A. Since the union of $A *_\varphi B$ and $A *_\psi C$ does not remove any arcs and coincides on all nodes of A including i_A and o_A it follows that the places i_A, o_A are the unique source and sink place of $A *_{(\varphi \cup \psi)} (B \cup C)$.
2. To prove that $A *_{(\varphi \cup \psi)} (B \cup C)$ is strongly connected, note that each note is in A, B or C and therefore in $A *_\varphi B$ or $A *_\psi C$. Since $A *_\varphi B$ and $A *_\psi C$ are C-nets and coincide on all nodes in A it follows directly that for any node x there is a directed path from i_A to x and x to o_A. □

We will also show that the vertical union is a C-net.

Lemma 5 (Vertical union of client-server compositions is a C-net). *Suppose that $A *_\varphi B$ and $B *_\phi C$ are client-server compositions of the C-nets A, B and C and the interfaces φ and ϕ and let $(P_A \cup P_B) \cap P_\varphi = \{i_B, o_B\}$ and $(P_B \cup P_C) \cap P_\phi = \{i_C, o_C\}$, then $A *_\varphi B \cup B *_\phi C$ is a C-net called the* vertical union.

Proof. $A *_\varphi B \cup B *_\phi C$ is a well defined labeled P/T-net. We now prove that $A *_\varphi B \cup B *_\phi C$ is a C-net.

1. Since $A *_\varphi B$ and $B *_\phi C$ intersect on the nodes of B it follows that i_A and o_A are the only two places with $\bullet i_A = o_A \bullet = \emptyset$.
2. To prove that $A *_\varphi (B *_\phi C)$ is strongly connected, we have to show that for any node x there is a directed path from i_A to x and there is a directed path from x to o_A. For nodes in $A *_\varphi B$ this follows since $A *_\varphi B$ is a C-net. For a node in $B *_\phi C$ this is also easy to see: Since $B *_\phi C$ is a C-net there are directed paths from i_B to x and from x to o_B. Since $A *_\varphi B$ is a C-net there are also directed paths from i_A to i_B and o_B to o_A. Combining the results yields that there are directed paths from i_A to x and x to o_A. □

We do not only want to consider horizontal and vertical unions of C-nets, but complete trees of C-nets. A *C-net tree* can be used to describe a complete system of components.

Definition 23 (C-net tree). *Let $\mathcal{C}$ be a set of C-nets and $\mathcal{I}$ be a set of interfaces. If the relation $R = \{(A, B) \in \mathcal{C} \times \mathcal{C} | \exists \phi \in \mathcal{I} \wedge A *_\phi B$ is a client-server composition with $\{i_B, o_B\} \in P_\phi\}$ has the structure of a tree, then $(\mathcal{C}, \mathcal{I})$ is called a* C-net tree.

By the lemma's 3, 4 and 5 it follows that a C-net tree and all of its sub-trees are also C-nets. Figure 1 is an example of a C-net tree. The set of C-nets $\mathcal{C}$ consists of the C-nets A, B, C, D, E, F, G, H, and I where A is the root C-net. The set of interfaces between the C-nets $\mathcal{I}$ is given by the arcs $\{(A, B), (A, E), (A, F), (B, C), (B, D), (F, G), (F, H), (F, I)\}$. There are two sub-trees in Figure 1: one consisting of the C-nets B, C and D where B is the root C-net and one consisting of the C-nets F, G, H and I where F is the root C-net. Clearly, due to the lemma's, the sub-trees and the complete tree are C-nets as well, symbolized by Y, Z and X respectively.
In this section we provided construction rules to build trees of C-nets. In the next section we will consider the behavior of these trees.

5 Construction consistency for C-nets

In this section we will provide compositionality results of the client-server compositions we introduced in the previous section. The compositionality results address the behavior of the composite C-net. It is important in software development that the behavior of a composition of a number of C-nets incorporates the behavior of the specifying root C-net. If that is the case, the system fulfils its requirements. We will formalize this quality requirement of a system by introducing the notion of *construction consistency*.

Definition 24 (Construction consistency). *Let $(\mathcal{C}, \mathcal{I})$ be a C-net tree. The tree is called* construction consistent *if and only if for each sub-tree with corresponding C-net X and root C-net A: $A \simeq_b X$, where all transitions in X except those in the root C-net A are relabeled to τ.*

In particular this definition is applicable to a single client-server composition. For example the C-net composed of the two C-nets in Figure 2 is construction consistent with *C-net 1* in the same figure: the two C-nets are glued together according to the rules of a client-server composition, *C-net 2* does not limit *C-net 1* in its behavior, and therefore the composite net has the same behavior as *C-net 1* if we relabel all transition in *C-net 2* to the τ-label.

Now we will prove two cases of construction consistency: the client net A has the same behavior as the nets $A *_{(\varphi \cup \psi)} \tau(B \cup C)$ and $A *_\varphi \tau(B *_\psi C)$ under the assumption that $A *_\varphi \tau(B)$ and $A *_\psi \tau(C)$ behave externally the same as A and that $B *_\psi \tau(C)$ behaves as B in terms of branching bisimulation. These compositionality results are key in constructing C-net trees. However they are not sufficient. Consider for

instance Figure 6. This figure depicts a horizontal union of two client-server compositions. $A *_\phi B$ is a composition of the C-nets A and B with interface ϕ and $A *_\psi C$ is a composition of the C-nets A and C with interface ψ. Both compositions are sound C-nets (Remark 1) given an initial marking of a single token in the source place. Moreover $A \simeq_b A *_\phi \tau(B)$ and $A \simeq_b A *_\psi \tau(C)$. However the horizontal union $A *_{(\phi \cup \psi)} (B \cup C)$ initially marked with a single token in the source place is not sound. This case is depicted in the figure. Component B is limiting the behavior of A: it forces the τ-labeled transition in the upper part

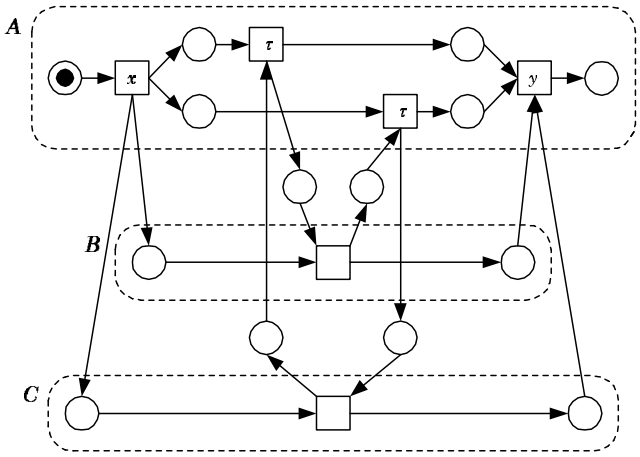

Fig. 6. Two sound client-server compositions and an unsound union.

of net A to be executed before the τ-labeled transition in the lower part. When we consider the behavior of $A *_\phi B$ this is not "noted" since the τ labels can not be distinguished from each other. Also component C is limiting the behavior of A and this is not also not "noted" for the same reason. However in the union of both client-server compositions the limitations result in a dead-lock of the process. Figure 6 is a motivation for not allowing τ-labeled transitions to be attached to the interface. If we replace in the example of Figure 6 the τ-labels by transitions with similar labels we find another erroneous example. In that case our conclusion is that transitions with the same label may not be attached differently to the interfaces.

After the compositionality theorems we will in addition show that the empty completion property can be transferred over branching bisimilar nets. This result enables us to construct systems with the empty completion option provided the root C-net has this option and the system is construction consistent. In the proofs of Theorem 1 and Theorem 2 we use that the related states $s_A \mathcal{R} s_U$ of the related process A and $A *_\phi B$ have the *projection property*, i.e., $s_U | P_A = s_A$. Lemma 6 shows that a projection branching bisimulation actually exists for compositions of C-nets where the transitions labels are unique. Note that in a client-server composition of A and B there are no arcs in $(A *_\phi B) \backslash A$ to places in P_A. Therefore, the connections between A to B can only limit the behavior of the A-part in $A *_\phi B$ and not extend it. Hence: If $s \in [A *_\phi B, [i_A^k]\rangle$ and $(A, s_A) \mathcal{R} (A *_\phi B, s)$, then $s_A \in [A, [i_A^k]\rangle$.

Lemma 6 (Projection branching bisimulation). *Let $A *_\varphi B$ be a client-server composition of the C-nets A and B and their interface φ and let $(P_A \cup P_B) \cap P_\varphi = \{i_B, o_B\}$. If*

1. *$\forall t \in T_A : \ell_A(t) \neq \tau$, i.e., all transitions in A have a visible label,*
2. *$\forall t_1, t_2 \in T_A$: if $\ell_A(t_1) = \ell_A(t_2)$, then $t_1 = t_2$, i.e., all labels are unique, and*
3. *$A \simeq_b A *_\varphi \tau(B)$,*

then for all branching bisimulations for C-nets $\mathcal{R}$, for all $k \in \mathbb{N}$ and for all $s \in [A *_\varphi B, [i_A^k]\rangle$ with $(A, s_A) \mathcal{R} (A *_\phi \tau(B), s)$ we have: $s | P_A = s_A$, i.e., the state s restricted to the net A is similar to the internal state s_A of A.

Proof. We denote $N = A *_\varphi \tau(B)$. Let $\mathcal{R}$ be an arbitrary branching bisimulation for C-nets and let k be an arbitrary number in $\mathbb{N}$. We need to prove that all related states are *projection* related. Since the initial states of A and N are equal (implied by Definition 19 and the first two requirements of this lemma), it follows that the projection property holds initially. Now suppose that for an arbitrary $s \in [N, [i_A^k]\rangle$ we have $(A, s_A)\mathcal{R}(N, s_N)$ with $s_N|P_A = s_A$. Then, since the label of t is unique (the first two requirements of this lemma), for all $t \in T_A$: If $(A, s_A)\ [t\rangle\ (A, s'_A)$, then $(N, s_N)\ [\ell_N(t)\rangle\ (N, s'_N)$ with $(A, s'_A)\mathcal{R}(N, s'_N)$. By the structure of net N it follows that $s'|P_A = s'_A$. Since s was an arbitrary reachable state, it follows that the projection property holds for any reachable state. □

In the next two theorems we require that the transition labels are unique and not equal to τ, similar to the first two requirement of Lemma 6. Lemma 6 then implies the existence of a *projection* branching bisimulation for C-nets. However if we would drop these requirements on uniqueness and the absence of τ-labeled transitions, but we would have a relation $\mathcal{R}$ that is a projection branching bisimulation for C-nets, then we would obtain the same results.

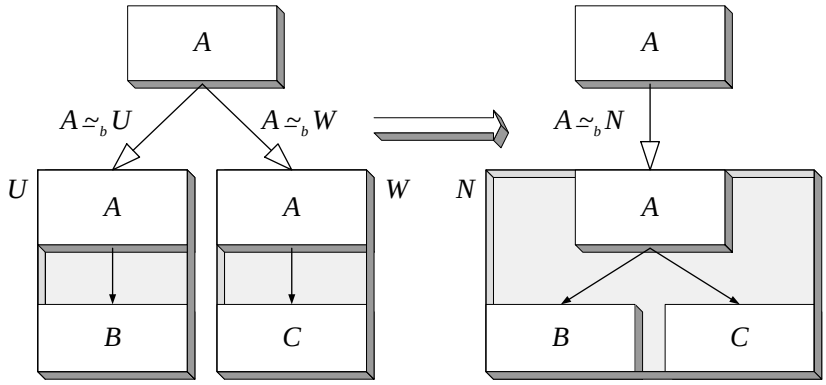

Fig. 7. The essence of Theorem 1.

Theorem 1 (Construction consistency of horizontal unions).
Let $A *_\varphi B$ and $A *_\psi C$ be client-server compositions of the C-nets A, B and C and their interfaces φ and ψ and let $(P_A \cup P_B) \cap P_\varphi = \{i_B, o_B\}$ and $(P_A \cup P_C) \cap P_\psi = \{i_C, o_C\}$. If

1. *$\forall t \in T_A : \ell_A(t) \neq \tau$, i.e., all transitions in A have a visible label,*
2. *$\forall t_1, t_2 \in T_A$: if $\ell_A(t_1) = \ell_A(t_2)$, then $t_1 = t_2$, i.e., all labels are unique,*
3. *$A \simeq_b A *_\varphi \tau(B)$ and $A \simeq_b A *_\psi \tau(C)$,*

then $A \simeq_b A *_{(\varphi\cup\psi)} \tau(B \cup C)$.

Proof. We denote $U = A *_\varphi \tau(B)$, $W = A *_\psi \tau(C)$, and $N = A *_{(\varphi\cup\psi)} \tau(B \cup C)$.
The states of the nets A, U, W, and N and the interfaces φ and ψ are bags. When we refer to a state s_x, s'_x, s''_x, then always $s_x, s'_x, s''_x \in \mathcal{B}(P_x)$ where one of the characters A, φ, ψ, U, W, or N is substituted for x. Since the interfaces φ and ψ also contain the source and sink places of the nets B and C, we will consider the states of the nets B and C *without* their source and sink places. When we refer to a state s_x, s'_x, s''_x, then we mean $s_x, s'_x, s''_x \in \mathcal{B}(P_x \backslash \{i_x, o_x\})$ where a character B or C is substituted for x.
In this proof we will use the relations between the behaviors of A and U and A and W to prove a relation between the behaviors of A and N.
$A \simeq_b U$ and $A \simeq_b W$ imply that there are branching bisimulations $\mathcal{R}_1$ and $\mathcal{R}_2$ that satisfy all properties of Definition 19 such that $\forall k \in \mathbb{N}$: $(A, [i_A^k])\ \mathcal{R}_1\ (U, [i_A^k])$ and $(A, [i_A^k])\ \mathcal{R}_2\ (W, [i_A^k])$.
By the construction of U such that $A \subseteq U$, the requirement on the uniqueness of the labels in A, and the fact that all related states are reachable from the initial state Lemma 6 implies that for related states s_A

and s_U the equality $s_U|P_A = s_A$ holds. Moreover it implies that if a transition t in A fires, then the same transition in U fires. Therefore if two states s_A and s_U are related, we may split up s_U in disjoint parts s_A, s_φ and s_B, namely $s_U = s_A + s_\varphi + s_B$. Since the same holds for A and W, we can also split up a state s_W in disjoint parts s_A, s_ψ and s_C, namely $s_W = s_A + s_\psi + s_C$. By Lemma 4 it follows that N is a C-net, and by construction we have that $U \subseteq N$, $W \subseteq N$ and $U \cap W = A$. The latter implies that we can also split up a state s_N in disjoint parts s_A, s_φ, s_B, s_ψ, s_C, namely $s_N = s_A + s_\varphi + s_B + s_\psi + s_C$.
Based on $\mathcal{R}_1$, $\mathcal{R}_2$, and the property to split the states as explained before, we define the relationship $\mathcal{R}$:

$$
\begin{aligned}
&(A, s_A)\mathcal{R}(N, s_N) \Leftrightarrow \\
&\exists s_\varphi, s_B, s_\psi, s_C : s_N = s_A + s_\varphi + s_B + s_\psi + s_C \wedge \\
&(A, s_A)\mathcal{R}_1(U, s_A + s_\varphi + s_B) \wedge (A, s_A)\mathcal{R}_2(W, s_A + s_\psi + s_C)
\end{aligned}
$$

We will first show that $\mathcal{R}$ is a branching bisimulation that satisfies the requirements of Definition 18. Consider markings s_A and s_N such that $(A, s_A)\ \mathcal{R}\ (N, s_N)$.

1. Assume that $t \in T_A$ and $(A, s_A)\ [\ell_A(t)\rangle\ (A, s'_A)$. We have to prove that there exists markings s'_N and s''_N such that

$$
\begin{aligned}
&(N, s_N) \Longrightarrow (N, {s_N}'')\ [\ell_N(t)\rangle\ (N, {s_N}')\wedge \\
&(A, s_A)\mathcal{R}(N, {s_N}'') \wedge (A, s'_A)\mathcal{R}(N, {s_N}').
\end{aligned}
$$

   Recall labels of transitions are unique, therefore we deal with the same transition t in U and N. The firing of $t \in T_A$ affects only places in P_A, P_φ and P_ψ and not the places in $P_B \backslash \{i_B, o_B\}$ and $P_C \backslash \{i_C, o_C\}$. This implies that the states in of B and C are unchanged when t fires ($s''_B = s'_B$ and $s''_C = s'_C$). Since $t \in T_A$ and all transition labels of transitions in A are visible we have $\ell_A(t) = \ell_U(t) = \ell_W(t) = \ell_N(t) \neq \tau$. Moreover the third requirement of this theorem implies that τ-steps are only taken in B and C ($s_A = s''_A$). From $(A, s_A)\mathcal{R}(N, s_N)$ and $(A, s_A)\ [\ell_A(t)\rangle\ (A, s'_A)$ and that $\mathcal{R}$ implies $\mathcal{R}_1$ it follows that there are s'_φ, s''_A, s''_φ, and s''_B such that

$$
\begin{aligned}
&(U, s_A + s_\varphi + s_B) \Longrightarrow \\
&(U, s_A + s''_\varphi + s''_B)\ [\ell_U(t)\rangle\ (U, s'_A + s'_\varphi + s''_B)
\end{aligned}
$$

   and

$$
\begin{aligned}
&(A, s_A)\mathcal{R}_1(U, s_A + s''_\varphi + s''_B)\wedge \\
&(A, s'_A)\mathcal{R}_1(U, s'_A + s'_\varphi + s''_B).
\end{aligned}
$$

   Analogously we find exactly the same expression for W where B is replaced by C and φ by ψ. Since the interfaces are disjoint, the τ-steps in B and C do not influence each other. Therefore we may execute them in arbitrary order as long as the orders in B and C are not mixed up. Moreover for the same reason the effects of the firing of t can be considered separately for each interface. Therefore we may combine these firing sequences:

$$
\begin{aligned}
&(N, s_A + s_\varphi + s_B + s_\psi + s_C) \Longrightarrow \\
&(N, s_A + s''_\varphi + s''_B + s''_\psi + s''_C)\ [\ell_N(t)\rangle\ (N, s'_A + s'_\varphi + s''_B + s'_\psi + s''_C).
\end{aligned}
$$

   According to the definition of $\mathcal{R}$ we have:

$$
\begin{aligned}
&(A, s_A)\mathcal{R}(N, s_A + s''_\varphi + s''_B + s''_\psi + s''_C)\wedge \\
&(A, s'_A)\mathcal{R}(N, s'_A + s'_\varphi + s''_B + s'_\psi + s''_C).
\end{aligned}
$$

   This concludes the proof of this part.

2. Assume now that $t \in T_N$ and $(N, s_A + s_\varphi + s_B + s_\psi + s_C)\ [\ell_N(t)\rangle\ (N, s'_A + s'_\varphi + s'_B + s'_\psi + s'_C)$. We can distinguish three different cases: $t \in T_A$, $t \in T_B$, or $t \in T_C$. In the first case $\ell_N(t) = \ell_U(t) = \ell_W(t) \neq \tau$ and in the last two cases $\ell_N(t) = \tau$. For each of these cases we need to prove that there exists markings s'_A and s''_A such that

$$(A, s_A) \Longrightarrow (A, {s_A}'')\ [\ell_A(t)\rangle\ (A, {s_A}')\wedge$$
$$(N, s_N)\mathcal{R}(N, {s_A}'') \wedge (N, s'_N)\mathcal{R}(A, {s_A}').$$

   (a) Suppose $t \in T_A$. Then $s_B = s'_B$ and $s_C = s'_C$. By the structure of the nets it follows that the same step can be made in U and W. Therefore we have

$$(U, s_A + s_\varphi + s_B)\ [\ell_U(t)\rangle\ (U, s'_A + s'_\varphi + s_B)\wedge$$
$$(W, s_A + s_\psi + s_C)\ [\ell_W(t)\rangle\ (W, s'_A + s'_\psi + s_C).$$

   By the properties of $\mathcal{R}_1$ and $\mathcal{R}_2$ it follows that

$$(A, s'_A)\mathcal{R}_1(U, s'_A + s'_\varphi + s_B)\wedge$$
$$(A, s'_A)\mathcal{R}_2(W, s'_A + s'_\psi + s_C).$$

   Hence by the definition of $\mathcal{R}$ this results in $(A, s'_A)\mathcal{R}(N, s'_A + s'_\varphi + s_B + s'_\psi + s_C)$. We do not need to consider τ-steps here because the steps in U and W are projections of the step in N.

   (b) Suppose $t \in T_B$. Then $\ell_U(t) = \tau$ and we have $s_A = s'_A$ since τ-transitions in B do not affect the state of A, and $s_\psi = s'_\psi$ and $s_C = s'_C$ since both the nets B and C and their interfaces are disjoint. By the properties of $\mathcal{R}_1$ and $\mathcal{R}_2$ we have

$$(U, s_A + s_\varphi + s_B)\ [\ell_U(t)\rangle\ (U, s_A + s'_\varphi + s'_B)\wedge$$
$$(A, s_A)\mathcal{R}_1(U, s_A + s'_\varphi + s'_B)\wedge$$
$$(A, s_A)\mathcal{R}_2(W, s_A + s_\psi + s_C).$$

   Hence, $(A, s_A)\mathcal{R}(N, s_A + s'_\varphi + s'_B + s_\psi + s_C)$.

   (c) For $t \in T_C$ a similar argument as for $t \in T_B$ holds where B is replaced by C and φ by ψ.

We now have proved that $\mathcal{R}$ is a branching bisimulation (Definition 18). In addition we have to prove that $\mathcal{R}$ is a branching bisimulation for C-nets which imposes additional constraints on the initial and final states as formulated in Definition 19. Note that since we required that all transitions labels are $\neq \tau$, it follows that the initial and final states are *uniquely* related; τ steps are not possible in this case. In the following existence and uniqueness proof we use similar arguments for the initial and final state. Therefore, for convenience, instead of writing down the same proof twice we using the character z which is a substitute for subsequently i and o.

1. We first prove existence.
   Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are branching bisimulations for C-nets we have $\forall k \in \mathbb{N}$:
   $(A, [z^k_A])\ \mathcal{R}_1(U, [z^k_A]) \wedge (A, [z^k_A])\mathcal{R}_2(W, [z^k_A])$.
   By the definition of $\mathcal{R}$, this implies that $\forall k \in \mathbb{N}$:
   $(A, [z^k_A])\mathcal{R}(N, [z^k_A])$,
2. Second we prove uniqueness, i.e., initial and final states are exclusively related to one another. $\forall k \in \mathbb{N}$: Suppose $(A, [z^k_A])\mathcal{R}(N, y)$. Then $y = [z^k_A]+s_\varphi+s_B+s_\psi+s_C$, but also $(A, [z^k_A])\mathcal{R}_1(U, [z^k_A]+s_\varphi+s_B)$ and $(A, [z^k_A])\mathcal{R}_2(W, [z^k_A] + s_\psi + s_C)$. Using the uniqueness properties of $\mathcal{R}_1$ and $\mathcal{R}_2$ this implies that $s_\varphi = s_B = s_\psi = s_C = \mathbf{0}$. Hence $y = [z^k_A]$.

We may now conclude that $A \simeq_b A *_{(\varphi\cup\psi)} \tau(B \cup C)$. □

It is easy to see that Theorem 1 can be extended to a horizontal unions that couples n ($n \in \mathbb{N}$) C-nets. Without proof we generalize the horizontal union in the following corollary.

Corollary 1 (Construction consistency of horizontal unions).
Let for $i = 1, \ldots, n$ $A *_{\varphi_i} B_i$ be client-server compositions of the C-nets A and B_i and their interfaces φ_i and let $(P_A \cup P_{B_i}) \cap P_{\varphi_i} = \{i_{B_i}, o_{B_i}\}$. If

1. *$\forall t \in T_A : \ell_A(t) \neq \tau$, i.e., all transitions in A have a visible label,*
2. *$\forall t_1, t_2 \in T_A$: if $\ell_A(t_1) = \ell_A(t_2)$, then $t_1 = t_2$, i.e., all labels are unique, and*
3. *$A \simeq_b A *_{\varphi} \tau(B_i)$ for $i = 1, \ldots, n$,*

then $A \simeq_b A *_{(\varphi_1 \cup \ldots \cup \varphi_n)} (B_1 \cup \ldots \cup B_n)$.

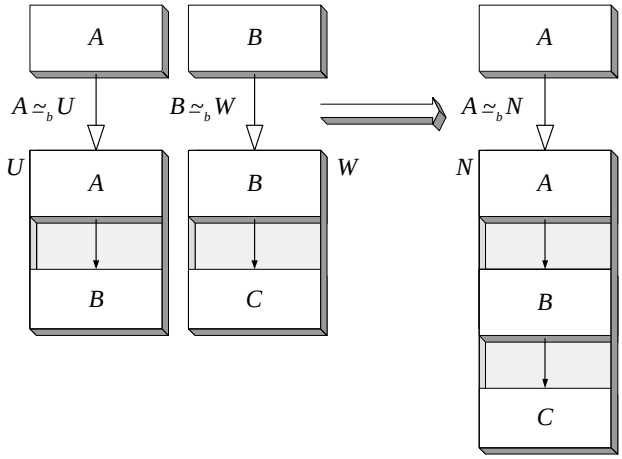

Fig. 8. The essence of Theorem 2.

Now we have derived a result for the horizontal extension of C-net trees. In the following theorem we will derive a result to extend a C-net tree vertically. With the combination of these results we are able to define the construction rules in the next section that enable us to build a complete tree given a number of client-server compositions.

Theorem 2 (Construction consistency of vertical unions).
Let $A *_{\varphi} B$ and $B *_{\psi} C$ be client-server compositions of the C-nets A, B and C and their interfaces φ and ψ and let $(P_A \cup P_B) \cap P_{\varphi} = \{i_B, o_B\}$ and $(P_B \cup P_C) \cap P_{\phi} = \{i_C, o_C\}$. If

1. *$\forall t \in T_A : \ell_A(t) \neq \tau$, i.e., all transitions in A have a visible label,*
2. *$\forall t \in T_B : \ell_B(t) \neq \tau$, i.e., all transitions in B have a visible label,*
3. *$\forall t_1, t_2 \in T_A$: if $\ell_A(t_1) = \ell_A(t_2)$, then $t_1 = t_2$, i.e., all labels are unique,*
4. *$\forall t_1, t_2 \in T_B$: if $\ell_B(t_1) = \ell_B(t_2)$, then $t_1 = t_2$, i.e., all labels are unique,*
5. *$A \simeq_b A *_{\varphi} \tau(B)$ and $B \simeq_b B *_{\psi} \tau(C)$,*

then $A \simeq_b A *_{\varphi} \tau(B *_{\psi} C)$.

Proof. This proof has similarities with the proof of Theorem 1, however to make it self-contained we repeat them. We use almost the same notations as in the proof of Theorem 1, except for the following differences: now $W = B *_{\psi} \tau(C)$ and $N = A *_{\varphi} \tau(B *_{\psi} C)$.
In this proof we will use the branching bisimulations between A and U and B and W to prove the branching bisimulation of A and N.
$A \simeq_b U$ and $B \simeq_b W$ imply that there are branching bisimulations $\mathcal{R}_1$ and $\mathcal{R}_2$ that satisfy all properties of Definition 19 such that $\forall k \in \mathbb{N}$: $(A, [i_A^k])\ \mathcal{R}_1\ (U, [i_A^k])$ and $(B, [i_A^k])\ \mathcal{R}_2\ (W, [i_A^k])$.
For the same reasons mentioned in the proof of Theorem 1 the conditions of Lemma 6 are satisfied. Therefore, again, if two states s_A and s_U are related, we may split up a state s_U in disjoint parts s_A, s_φ and s_B.

Since the same holds for B and W, we can also split up a state s_W in disjoint parts s_B, s_ψ and s_C. By Lemma 4 it follows that N is a C-net, and by construction we have that $U \subseteq N$, $W \subseteq N$ and $U \cap W = B$. The latter implies that we can also split up a state s_N in disjoint parts s_A, s_φ, s_B, s_ψ, s_C.
Based on $\mathcal{R}_1$, $\mathcal{R}_2$, we define the relationship $\mathcal{R}$:

$$
\begin{aligned}
&(A, s_A)\mathcal{R}(N, s_N) \Leftrightarrow \\
&\exists s_\varphi, s_B, s_\psi, s_C : s_N = s_A + s_\varphi + s_B + s_\psi + s_C \wedge \\
&(A, s_A)\mathcal{R}_1(U, s_A + s_\varphi + s_B) \wedge (B, s_B)\mathcal{R}_2(W, s_B + s_\psi + s_C).
\end{aligned}
$$

We will first show that $\mathcal{R}$ is a branching bisimilarity by verifying the two requirements of Definition 18 and second we will show that $\mathcal{R}$ satisfies the additional requirements for C-nets.
Consider two markings s_A and $s_N = s_A + s_\varphi + s_B + s_\psi + s_C$ such that $(A, s_A)\ \mathcal{R}\ (N, s_N)$.

1. Assume that $t \in T_A$ and $(A, s_A)\ [\ell_A(t)\rangle\ (A, s'_A)$. We have to prove that there exists markings s'_N and s''_N such that

$$
\begin{aligned}
&(N, s_N) \Longrightarrow (N, {s_N}'')\ [\ell_N(t)\rangle\ (N, {s_N}')\wedge \\
&(A, s_A)\mathcal{R}(N, {s_N}'') \wedge (A, s'_A)\mathcal{R}(N, {s_N}').
\end{aligned}
$$

   The firing of $t \in T_A$ only affects places in P_A and P_φ. This implies that the firing of t does not affect the state in B ($s_B = s'_B$), C ($s_C = s'_C$) and ψ ($s_\psi = s'_\psi$). Since $t \in T_A$ and we assumed that all transition labels of transitions in A in U are visible we have $\ell_A(t) = \ell_U(t) \neq \tau$. Moreover in U τ-steps are only taken in B. From $(A, s_A)\mathcal{R}(N, s_N)$ and since $\mathcal{R}$ implies $\mathcal{R}_1$, it follows that $(A, s_A)\mathcal{R}_1(U, s_A + s_\varphi + s_B)$. From $(A, s_A)\ [\ell_A(t)\rangle\ (A, s'_A)$ and $\mathcal{R}_1$ it follows that

$$
\begin{aligned}
&(U, s_A + s_\varphi + s_B) \Longrightarrow (U, s_A + s''_\varphi + s''_B)\ [\ell_U(t)\rangle\ (U, s'_A + s'_\varphi + s''_B)\wedge \\
&(A, s_A)\mathcal{R}_1(U, s_A + s''_\varphi + s''_B) \wedge (A, s'_A)\mathcal{R}_1(U, s'_A + s'_\varphi + s''_B). \qquad \textbf{(A)}
\end{aligned}
$$

   This implies that there is a firing sequence of transitions $t_{B_1}, \ldots, t_{B_n}$ in T_B such that

$$
\begin{aligned}
&(U, s_A + s_\varphi + s_B)\ [\ell_U(t_{B_1})\rangle\ (U, s_A + s_{\varphi_1} + s_{B_1})[\ell_U(t_{B_2})\rangle \\
&\ldots \\
&(U, s_A + s_{\varphi_{n-1}} + s_{B_{n-1}})\ [\ell_U(t_{B_n})\rangle\ (U, s_A + s''_\varphi + s''_B). \qquad \textbf{(B)}
\end{aligned}
$$

   See Figure 9(a). By the requirements of the branching bisimulation definition for C-nets it follows that, if $s_B = \mathbf{0}$, then $s_B + s_\psi + s_C = \mathbf{0}$. In this case it follows directly that $s'_A \mathcal{R} s'_A + s'_\varphi + s_B + s_\psi + s_C$. Suppose that $s_B \neq \mathbf{0}$. By the definition of $\mathcal{R}$ it follows that the states s_B in B and $s_B + s_\psi + s_C$ in W are related. [4]
   Using the definition of $\mathcal{R}_2$ it follows that there exists a row of markings $s_B + s_\psi + s_C, \ldots, s_{B_{n-1}} + s_{\psi_{n-1}} + s_{C_{n-1}}$, $s''_B + s''_\psi + s''_C$ as indicated in Figure 9(b). Since in W τ-steps only occur in C and

[4] For the readers that are familiar with the compositionality theorem in [9]: By the branching bisimulation relation for C-nets of Definition 18 it follows that also in case of multiple entrance *all* states are related. Therefore we do not need the *safeness* and *activation safeness* requirements we used in [9] to ensure single entrance, which makes analysis less complicated. However the class of nets which satisfy the stronger branching bisimilarity requirements of Definition 18 is smaller.

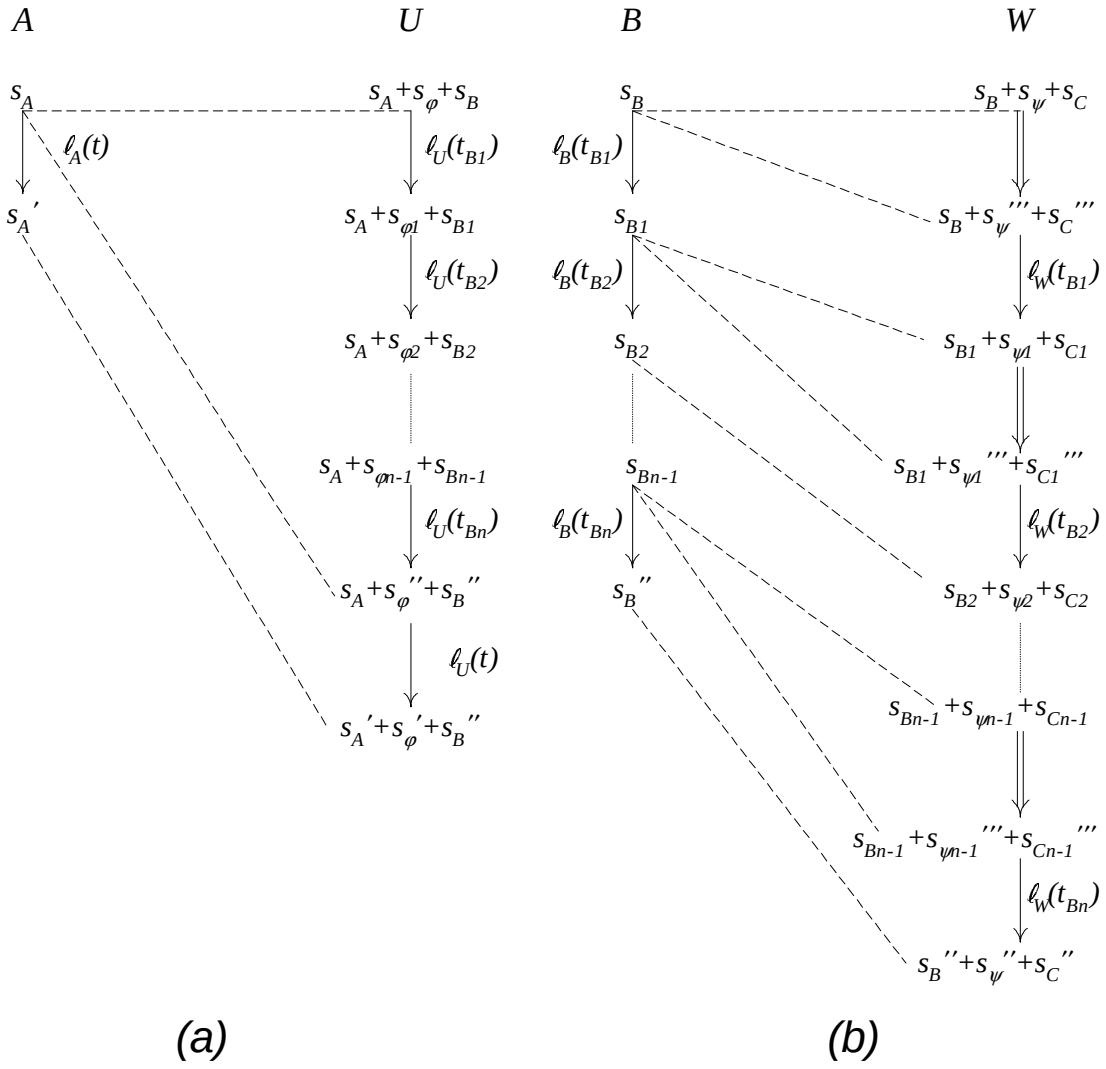

Fig. 9. Vertical union of branching bisimilar components.

firings in B do not affect places in C and vice versa firings in C do not affect places in B, we have:

$$
\begin{array}{l}
(W, s_B + s_\psi + s_C) \\
\Longrightarrow (W, s_B + s'''_\psi + s'''_C)\ [\ell_W(t_{B_1})\rangle\ (W, s_{B_1} + s_{\psi_1} + s_{C_1}) \\
\Longrightarrow (W, s_{B_1} + s'''_{\psi_1} + s'''_{C_1})\ [\ell_W(t_{B_2})\rangle\ (W, s_{B_2} + s_{\psi_2} + s_{C_2}) \\
\ldots \\
\Longrightarrow (W, s_{B_{n-1}} + s'''_{\psi_{n-1}} + s'''_{C_{n-1}})\ [\ell_W(t_{B_n})\rangle\ (W, s''_B + s''_\psi + s''_C) \wedge \\
s'''_C = s_{C_1}, \text{for all } i \in \{1, \ldots, n-1\} : s'''_{C_i} = s_{C_{i+1}}, s'''_{C_{n-1}} = s''_C \wedge \\
(B, s_B) \mathcal{R}_2 (W, s_B + s'''_\psi + s'''_C) \wedge \\
\text{for all } i \in \{1, \ldots, n-1\} : (B, s_{B_i}) \mathcal{R}_2 (W, s_{B_i} + s_{\psi_i} + s_{C_i}) \wedge \\
\text{for all } i \in \{1, \ldots, n-1\} : (B, s_{B_i}) \mathcal{R}_2 (W, s_{B_i} + s'''_{\psi_i} + s'''_{C_i}) \wedge \\
(B, s''_B) \mathcal{R}_2 (W, s''_B + s''_\psi + s''_C). \qquad \textbf{(C)}
\end{array}
$$

Composition of (**A**), (**B**) and (**C**) is possible since all states reached in B are related states in $\mathcal{R}_2$. Composition yields:

$$
\begin{array}{l}
(N, s_A + s_\varphi + s_B + s_\psi + s_C) \Longrightarrow \\
(N, s_A + s_\varphi + s_B + s'''_\psi + s'''_C)[\ell_N(t_{B_1})\rangle \\
(N, s_A + s_{\varphi_1} + s_{B_1} + s_{\psi_1} + s_{C_1}) \Longrightarrow \\
(N, s_A + s_{\varphi_1} + s_{B_1} + s'''_{\psi_1} + s'''_{C_1})[\ell_N(t_{B_2})\rangle \\
(N, s_A + s_{\varphi_2} + s_{B_2} + s_{\psi_2} + s_{C_2}) \\
\ldots \Longrightarrow \\
(N, s_A + s_{\varphi_{n-1}} + s_{B_{n-1}} + s'''_{\psi_{n-1}} + s'''_{C_{n-1}})[\ell_N(t_{B_n})\rangle \\
(N, s_A + s''_\varphi + s''_B + s''_\psi + s''_C)[\ell_N(t)\rangle \\
(N, s'_A + s'_\varphi + s''_B + s''_\psi + s''_C) \wedge \\
\text{recall, also here:} \\
s'''_C = s_{C_1}, \text{for all } i \in \{1, \ldots, n-1\} : s'''_{C_i} = s_{C_{i+1}}, s'''_{C_{n-1}} = s''_C.
\end{array}
$$

In N $\ell_N(t_{B_i}) = \tau$ $(i = 1, \ldots, n)$ and also all firings in C are τ-steps. Hence

$$
\begin{array}{l}
(N, s_A + s_\varphi + s_B + s_\psi + s_C) \Longrightarrow \\
(N, s_A + s''_\varphi + s''_B + s''_\psi + s''_C)[\ell_N(t)\rangle \\
(N, s'_A + s'_\varphi + s''_B + s''_\psi + s''_C).
\end{array}
$$

We now summarize the key-relations we found in (**A**) and (**C**):
(a) $(A, s_A) \mathcal{R}_1 (U, s_A + s''_\varphi + s''_B) \wedge$
(b) $(A, s'_A) \mathcal{R}_1 (U, s'_A + s'_\varphi + s''_B) \wedge$
(c) $(B, s''_B) \mathcal{R}_2 (W, s''_B + s''_\psi + s''_C)$.

By the definition of $\mathcal{R}$ and by combining:
- (a) and (c) yields $(A, s_A) \mathcal{R} (N, s_A + s''_\varphi + s''_B + s''_\psi + s''_C)$ and
- (b) and (c) yields $(A, s'_A) \mathcal{R} (N, s'_A + s''_\varphi + s''_B + s''_\psi + s''_C)$.

This proves one side of the branching bisimulation.

2. Let now be given that $(A, s_A) \mathcal{R} (N, s_A + s_\varphi + s_B + s_\psi + s_C)$. By the definition of $\mathcal{R}$ it follows that $(A, s_A) \mathcal{R}_1 (N, s_A + s_\varphi + s_B)$ and $(B, s_B) \mathcal{R}_2 (N, s_B + s_\psi + s_C)$. We now take a step in net N. Assume that $(N, s_A + s_\varphi + s_B + s_\psi + s_C)\ [\ell_N(t)\rangle\ (N, s'_A + s'_\varphi + s'_B + s'_\psi + s'_C)$. We can distinguish three

different cases: $t \in T_A$, $t \in T_B$, or $t \in T_C$. For each of these cases we need to prove that there exists markings s'_A and s''_A such that

$$(A, s_A) \Longrightarrow (A, s_A{}'')\ [\ell_A(t)\rangle\ (A, s_A{}')\wedge$$
$$(N, s_N)\mathcal{R}(A, s_A{}'') \wedge (N, s'_N)\mathcal{R}(A, s_A{}').$$

(a) Suppose $t \in T_A$. It follows that $s_B = s'_B$, $s_\psi = s'_\psi$ and $s_C = s'_C$. By the structure of the nets the same step can be made in U and therefore we have $(U, s_A + s_\varphi + s_B)\ [\ell_U(t)\rangle\ (U, s'_A + s'_\varphi + s_B)$. Since no τ-steps can be made in A we have $\ell_U(t) \neq \tau$ and hence $(A, s'_A)\mathcal{R}_1(N, s'_A + s'_\varphi + s_B)$. Using the relation $\mathcal{R}_2$ this adds to $(A, s'_A)\mathcal{R}(N, s'_A + s'_\varphi + s_B + s_\psi + s_C)$.

(b) Suppose $t \in T_B$. Then $s_A = s'_A$ and $s_C = s'_C$. By the structure of the nets it follows that the transition can be made as well in U and in W. With respect to U t is a τ-step and with respect to W t is not a τ-step. Hence $\ell_U(t) = \tau$ and $\ell_W(t) \neq \tau$ and so $(U, s_A{+}s_\varphi{+}s_B)[\ell_U(t)\rangle(U, s_A{+}s'_\varphi{+}s'_B)$ and $(W, s_B{+}s_\psi{+}s_C)[\ell_W(t)\rangle(W, s'_B{+}s'_\psi{+}s_C)$. From $\mathcal{R}_1$ and the fact that no τ-steps are possible in A this yields $(A, s_A)\mathcal{R}_1(U, s_A{+}s'_\varphi{+}s'_B)$ and from $\mathcal{R}_2$ this yields $(B, s'_B)\mathcal{R}_2(W, s'_B{+}s'_\psi{+}s_C)$. Adding these two relations gives $(A, s_A)\mathcal{R}(N, s_A + s'_\varphi + s'_B + s'_\psi + s_C)$.

(c) Finally suppose $t \in T_C$. Then $s_A = s'_A$, $s_\varphi = s'_\varphi$, and $s_B = s'_B$. By the structure of the nets it follows (projection) that $(W, s_B + s_\psi + s_C)\ [\ell_W(t)\rangle\ (W, s_B + s'_\psi + s'_C)$. By $\mathcal{R}_2$ and the fact that τ-steps in W occur in C we have that $(B, s_B)\mathcal{R}_2(W, s_B + s'_\psi + s'_C)$. Using $\mathcal{R}_1$ this adds to $(A, s_A)\mathcal{R}(N, s_A + s_\varphi + s_B + s'_\psi + s'_C)$.

We now have proved that $\mathcal{R}$ is a branching bisimulation. In addition we have to prove that $\mathcal{R}$ satisfies the additional requirements for C-nets which imposes additional constraints on the initial and final states. We will use similar arguments for the initial and final state as we used in the proof of Theorem 1. For convenience, instead of writing down the same proof twice we using the character z which is a substitute for subsequently i and o.

1. We first prove existence. Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are branching bisimulations for C-nets we have
   $\forall k \in \mathbb{N} : (A, [z_A^k])\ \mathcal{R}_1(U, [z_A^k])$ and
   $\forall l \in \mathbb{N} : (B, [z_B^l])\ \mathcal{R}_2(W, [z_B^l])$.
   Hence for $l = 0$, i.e., the empty net:
   $\forall k \in \mathbb{N} : (A, [z_A^k])\ \mathcal{R}(N, [z_A^k])$.
2. Second we prove uniqueness, i.e., initial and final states are exclusively related to one another. We have to prove:
   $\forall k \in \mathbb{N} : \forall y : (A, [z_A^k])\mathcal{R}(N, y) \Rightarrow y = [z_A^k]$.
   By definition we have
   $\forall k \in \mathbb{N} : (A, [z_A^k])\mathcal{R}_1(N, s_A + s_\varphi + s_B)$. This implies $s_\varphi + s_B = \mathbf{0}$ and $s_A = [z_A^k]$. Hence $y = [z_A^k]$.
   We only have to verify that $(B, \mathbf{0})\mathcal{R}_2(W, \mathbf{0})$. This follows from taking $k = 0$ in the definition of $\mathcal{R}_2$.
   Adding $\mathcal{R}_1$ and $\mathcal{R}_2$ gives the desired result.

Hence $A \simeq_b A *_\varphi \tau(B *_\psi C)$. □

The theorem for horizontal unions can be extended to an arbitrary number of nets (Corollary 1). In the vertical union the net C may be a result of unions as well. As a consequence multiple application of the branching bisimilarity theorems is possible and construction consistency for a C-net tree as long as each node and all of its child-nodes (and grand-child-nodes) is a client-server construction satisfying the conditions of one of the branching bisimilarity theorems. For an example consider Figure 10. If $A \simeq_b A *_\phi \tau(B)$, $A \simeq_b A *_\psi \tau(C)$, $B \simeq_b B *_\varphi \tau(D)$, and $B \simeq_b B *_\vartheta \tau(E)$, then $B \simeq_b B *_{\varphi\cup\vartheta} \tau(D \cup E)$ and $A \simeq_b A *_{\phi\cup\psi} \tau(B \cup C)$, and hence, $A \simeq_b A *_{\phi\cup\psi} \tau((B *_{\varphi\cup\vartheta} (D \cup E)) \cup C)$. The construction consistency theorems can be extended to more general theorems that allow in the client C-nets the use of transitions with non-unique or τ-labels. In fact we claim that we may drop the first two requirements of Theorem 1 and Corollary 1 and the first four requirements of Theorem 2. Instead to prove construction consistency, for Theorem 1, we should impose the following weaker requirements:

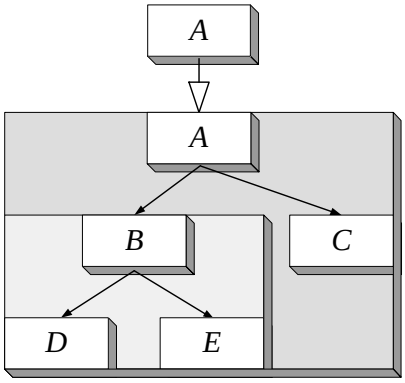

Fig. 10. A tree of coupled C-nets with the same observable behavior as the net A.

1. $\forall t \in T_A : (\ell_A(t) = \tau) \Rightarrow ((\bullet t \cup t\bullet) \cap (P_\phi \cup P_\psi) = \emptyset)$, i.e., τ-labeled transitions are not connected to the interface,
2. $\forall t_1, t_2 \in T_A : (\ell_A(t_1) = \ell_A(t_2)) \Rightarrow ((P_\phi \cup P_\psi) \cap \bullet t_1 = (P_\phi \cup P_\psi) \cap \bullet t_2 \wedge (P_\phi \cup P_\psi) \cap t_1\bullet = (P_\phi \cup P_\psi) \cap t_2\bullet)$, i.e., transition with the same label are similarly connected to the interface.

For Corollary 1 and Theorem 2 we could also adjust the requirements in a similar way.

Many properties are transferred from one system to another if they are branching bisimilar [43]. Here we show that the empty completion property transfers between branching bisimilar nets.

Lemma 7 (Transfer of empty completion property for branching bisimilar nets). *Let A and B be C-nets. Suppose that all transitions in A and B have a unique label except for the τ label and that $A \simeq_b B$. $\forall k \in \mathbb{N} : (\forall s : s \in [A, [i_A^k]\rangle \Rightarrow [o_A^k] \in [A, s)\rangle \Rightarrow (\forall s : s \in [B, [i_B^k]\rangle \Rightarrow [o_B^k] \in [B, s)\rangle$, i.e., if A satisfies the empty completion property, then B also satisfies the empty completion property.*

Proof. Pick an arbitrary $k \in \mathbb{N}$ and suppose we have a marking $s_B \in [B, [i_B^k]\rangle$. There is a firing sequence σ_B such that $(B, [i_B^k])\ [\sigma_B\rangle (B, s_B)$. Since $(A, [i_A^k]) \sim_b (B, [i_B^k])$ there is a marking $s_A \in [A, [i_A^k]\rangle$ and, since all transition labels are unique, there is a firing sequence σ_A such that $(A, [i_A^k])\ [\sigma_A\rangle\ (A, s_A)$ and $(A, s_A) \sim_b (B, s_B)$. If we remove all τ-transitions from the firing sequences σ_A and σ_B, then we have $\sigma_A = \sigma_B$. Clearly since A satisfies the k-empty completion property there is a firing sequence σ'_A such that $(A, s_A)\ [\sigma'_A\rangle\ (A, [o_A^k])$. Using again the branching bisimilarity, there is also a firing sequence σ'_B such that $(B, s_B)\ [\sigma'_B\rangle\ (B, x_B)$ with $[o_A^k] \sim_b x_B$. Since $[o_A^k]$ is the end state, and end states are uniquely related, also x should be the end state and so $x = [o_B^k]$. Hence B also satisfies the k-empty completion property. Since k was arbitrarily chosen we may conclude that B satisfies the empty completion property. □

An important consequence of this lemma is the following: When a C-net tree has the same visible behavior as its root-net and the root-net has the empty completion option, then the complete tree the empty completion option. Considering again Figure 10 this implies that, if A has the proper completion property, then $A *_{\phi\cup\psi} \tau((B *_{\varphi\cup\vartheta} (D \cup E)) \cup C)$ also has the proper completion property. This is a desirable property in software design. Intuitively it means that we can transfer the correctness of the specification to the implementation as long as both have the same visible behavior.

6 A design pattern for C-net trees

In this section we will introduce one design pattern to create trees of C-nets as presented in Figure 1. In future publication we hope to extend the number of design patterns. A C-net itself can be created in a constructive manner [1] as well. In [8, 26] building-blocks are used to create nets with the theory of graph grammars. Under certain assumptions these construction rules also apply to create C-nets. However the

creation of C-nets is not the subject of this paper. Once we have a number of C-nets, we are equipped with the proper building blocks to create a tree of C-nets. To create a client-server composition we introduced the operator "*". This composition is not necessarily construction consistent. We can think of many examples where an extension of one C-net with another would limit the behavior of the specification. To facilitate the consistency requirement we try to find a pattern to extend a C-net A with another C-net B in a consistent manner, i.e., the behavior of the composite net $A *_\varphi \tau(B)$ is branching bisimilar to the behavior of the initial C-net A in isolation. To limit the number of possibilities we assume that the extension, i.e., net B, has the proper completion property and that it is only connected with its source and sink places to the base net A, hence for the interface φ we have $P_\varphi = \{i_B, o_B\}$. Even under these assumptions it is not for granted that the composition $A *_\varphi B$ is construction consistent with the initial net A. In Figure 11 net A is the base net and net B is the extension of A. In all three presented patterns both A and B have the proper completion property in isolation. We consider the case in which there is initially one token in the source place.
In Figure 11(a) in addition the transitions that produce a token for net B and consume a token from net B are on a path from i_A to o_A. However if the production to B is skipped, then it is also not possible to fire the transition that consumes from B. Moreover if the transition fires which produces to B followed by the transition which does not consume from B, then a token is left in the sink places of B. Clearly the coupling with B influences the behavior of A.
In Figure 11(b) another problem occurs. The number of iterations in the first loop might not be equal to the number of iterations in the second loop, causing to more tokens produced to B then consumed from B, or a dead-lock in A. Clearly also this coupling influences the behavior of A.
Finally, in Figure 11(c), we consider an example in which A is acyclic in an attempt to prevent the errors in the second example. However this example is not safe. In net A the first transition produces three tokens: one in the second place, one in the third place and one in the top place. At that moment the second and the third transition are enabled. When the second transition fires once and the third transition fires twice we are in the situation that there is still one token in the top place and there are two tokens in fourth sequential place. In spite of the fact that the fourth place is marked with two tokens and an input place of the fourth transition, this transition can only fire once since there is only one token in the top place. If this transition fires, the last transition will be enabled with precisely one token in both of its input places. Hence A has the proper completion option in isolation. When we consider the composite net we see that one token is put into the source place i_B of B. However the behavior of A is such that two tokens should be consumed from B. Clearly the transition attached to o_B can only fire once and hence this results in a dead-lock. So we found another example where the coupling of A and B influences the behavior of A.
All examples have in common that the number of productions to B is not equal to the number of consumptions from B. This drawback is resolved in the following theorem.

Theorem 3 (Request-response pattern). *Let $A *_\varphi B$ be a client-server composition of C-nets A and B and the interface φ and let $(P_A \cup P_B) \cap P_\varphi = \{i_B, o_B\}$. Suppose that B has the proper completion property and suppose that there is a C-net $C \subseteq A$ with $F_C = F_A \cap ((P_C \times T_C) \cup (T_C \times P_C))$ and the proper completion property. See Figure 12. If*

1. *all arcs from $A \backslash C$ to C end in i_C and all arcs from C to $A \backslash C$ start in o_C, and*
2. *$\forall t \in start(C) : i_B \in t\bullet \wedge \forall t \in stop(C) : o_B \in \bullet t$,*

then $A \sim_b A *_\varphi \tau(B)$.

Proof. The proper completion property of C implies that, separated from its environment, the sum of all the firings of all the transitions in the set $start(C)$ equals the sum of all the firings of all the transitions in the set $stop(C)$.
Now we consider C in its environment. There are four types of connections to $A \backslash C$ and B:

1. the arcs from transitions in $A \backslash C$ to i_C,
2. the arcs from the transitions in $start(C)$ to i_B,
3. the arcs from o_B to the transitions in $stop(C)$, and

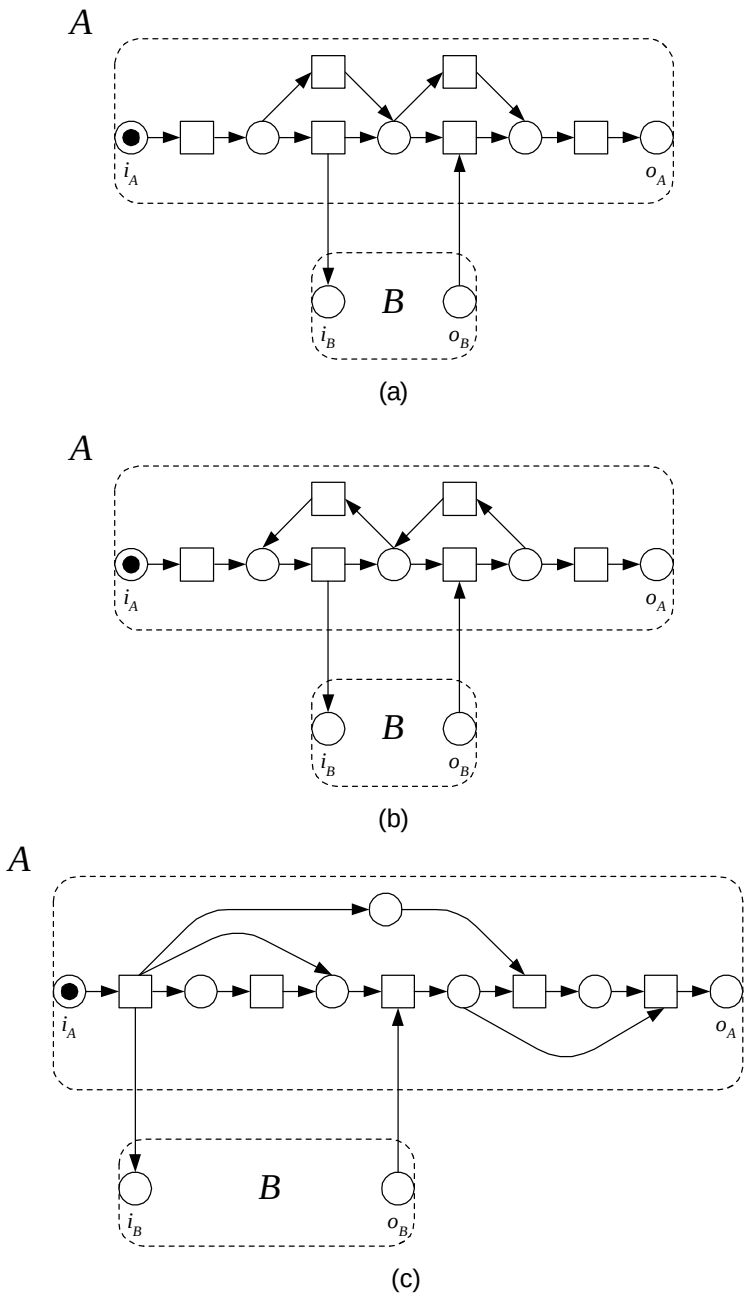

Fig. 11. Erroneous simple-structured patterns.

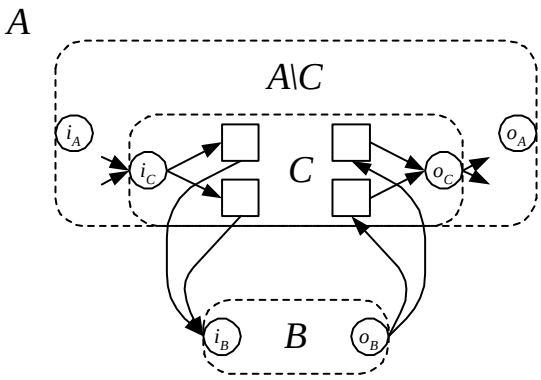

Fig. 12. Request-response pattern.

4. the arcs from $stop(C)$ to the transitions in $A\backslash C$.

Type 1 and 4 imply that the behavior of C, once a token has been places in i_C, can never be influenced by tokens in the net $A\backslash C$. Hence the only influences on the behavior of A can be raised by B.
Let us consider B. This net has the proper completion property in isolation, but also in its environment this property holds since its only connections with C are the type 2 and 3 connections.
Since all transitions in $start(C)$ are connected to i_B, this implies that the number of firings of all transitions in i_B equals the number of tokens that comes across o_B. Since all transitions in $stop(C)$ are connected to o_B, this implies that for any firing of a transition in $stop(C)$ there is a tokens available in o_B and that there will be no tokens left in B.
This implies that B at most postpones the behavior of C but never limits this behavior. □

By means of this request-response pattern we are able to build construction consistent client-server compositions of C-nets without having to verify the branching bisimilarity. A direct consequence of this theorem and Lemma 7 is that, if A has the empty completion property, then also $A *_\varphi B$ has the empty completion property. More general: If a root C-net of a C-net tree has the proper completion option and the tree is constructed by repeatedly applying the pattern, then the C-net of the whole tree has the empty completion option. This may be compared to the *inheritance-preserving transformation rules* to *construct* sub classes of net presented in [4, 14].

The pattern presented in this section fits in an approach to process design which is applied occasionally. In the first step a rough process is designed, then a more detailed specification follows and finally in the third step, i.e., the implementation, various applications have to execute the transitions. Consider for example Figure 13. It illustrates this approach with the decomposition of a transition a. In the example depicted in Figure 13 in the first step transition a is lifted out of the rough process. In the second step this transition is decomposed in two sub-transitions: a' is used to start with the activities and the original a label is used to mark the completion of the transition. The two transition are part of the client process. In the third step a server process is designed that actually executes some of the activities that should be performed by a. The server process is attached to a' and a by using the design pattern. If necessary also transitions in the server process can be decomposed in a similar way. If that happens, instead of the two layer of this example, we will end up with a C-net tree.

7 Conclusion

The contribution of this paper is a more efficient approach towards designing construction consistent software architectures. Whereas we were used to make constructions and execute exhaustive verifications of the state space afterwards and if necessary correct the constructions, we now have “correctness by construction”. We demonstrated that we can build an entire tree of C-net by starting with a single component and

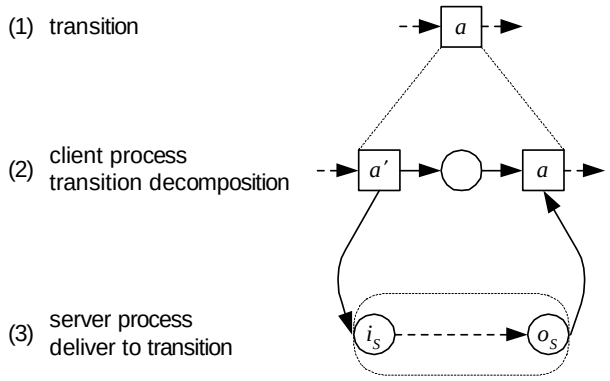

Fig. 13. An approach to process design using the pattern.

step by step adding new components such that the complete system of components has the same visible behavior as its specifying root component. We introduced an operator to compose C-nets and we presented new compositionality results for this operator that at the same time prescribed the conditions for consistent composition. Finally we introduced an easy way to satisfy the conditions of the compositionality theorems by presenting a design pattern. Using the pattern to build C-nets trees guaranteed consistency.

A second result is that we showed that our construction rules conserve the empty completion property, i.e., be able to finish a transaction and to finish without leaving any tokens. In the context of software development this yields that if specification has this properly, then also the complete system has this property.

We conclude by pointing out our future investigations. In this paper we presented "correctness by construction" as an alternative for a posteriori verification (i.e., "model checking"). In the future we will continue investigating this new course by constructing more sophisticated patterns to couple C-nets.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.
4. W.M.P. van der Aalst and T. Basten. Life-cycle Inheritance: A Petri-net-based Approach. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 62–81. Springer-Verlag, Berlin, 1997.
5. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.
6. W.M.P. van der Aalst, P. de Crom, R. Goverde, K.M. van Hee, W. Hofman, H. Reijers, and R.A. van der Toorn. ExSpect 6.4: An Executable Specification Tool for Hierarchical Colored Petri Nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 455–464. Springer-Verlag, Berlin, 2000.
7. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
8. W.M.P. van der Aalst, K.M. van Hee, and H.A. Reijers. Analysis of Discrete-time Stochastic Petri Nets. *Statistica Neerlandica*, 54(2):237–255, 2000.
9. W.M.P. van der Aalst, K.M. van Hee, and R.A. van der Toorn. Component-Based Software Architectures: A Framework Based on Inheritance of Behavior. *Science of Computer Programming*, 42(2-3):129–171, 2002.
10. W.M.P. van der Aalst, K.M. van Hee, and R.A. van der Toorn. Erratum to Compositionality of Projection Inheritance. *Science of Computer Programming*, 44(3):343–344, 2002.

11. C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*. Oxford University Press, New York City, 1977.
12. P. America. Designing an Object-Oriented Programming Language with Behavioral Subtyping. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Foundation of Object-Oriented Languages*, volume 489 of *Lecture Notes in Computer Science*, pages 60–90. Springer-Verlag, Berlin, 1991.
13. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Series in Software Engineering. Addison Wesley, Reading, MA, USA, 1998.
14. T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1998.
15. T. Basten and W.M.P. van der Aalst. Inheritance of Behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
16. R. Bastide and P. Palanque et al. Petri-Net Based Behavioural Specification of CORBA Systems. In *Application and Theory of Petri Nets 1999*, volume 1639 of *Lecture Notes in Computer Science*, pages 66–85. Springer-Verlag, Berlin, 1999.
17. IEEE-SA Standards Board. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Technical report, IEEE New York, October 2000. IEEE Std 1471-2000.
18. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, Reading, MA, USA, 1998.
19. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern Oriented Software Architecture: A system of Patterns*. John Wiley and Sons, New York, 1996.
20. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
21. E.W. Dijkstra. The Structure of the 'T.H.E.' Multiprogramming System. *Communications of the ACM*, 18(8):453–457, 1968.
22. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison Wesley, Reading, MA, USA, 1995.
23. D. Garlan, R.T. Monroe, and D. Wile. Acme: An Architecture Description Interchange Language. In *Proceedings of CASCON'97*, pages 169–183, Toronto, Ontario, November 1997.
24. D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pages 1–39, Singapore, 1993. World Scientific Publishing Company.
25. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
26. G.Lei. *Graph Grammars and Petri Nets with References*. PhD thesis, University of Leiden, Leiden, Netherlands, 1990.
27. K.M. van Hee. *Information System Engineering: a Formal Approach*. Cambridge University Press, 1994.
28. K.M. van Hee, R.A. van der Toorn, J. van der Woude, and P. Verkoulen. A Framework for Component Based Software Architectures. In W.M.P. van der Aalst, J. Desel, and R. Kaschek, editors, *Software Architectures for Business Process Management (SABPM'99)*, pages 1–20, Heidelberg, Germany, June 1999. Forschungsbericht Nr. 390, University of Karlsruhe, Institut AIFB, Karlsruhe, Germany.
29. H. Kilov and W. Harvey, editors. *Object-Oriented Behavioral Specifications*, volume 371 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, MA, USA, 1996.
30. H. Kilov, B. Rumpe, and I. Simmonds, editors. *Behavioral Specifications of Businesses and Systems*, volume 523 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, MA, USA, 1999.
31. M. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, and H. Lipson. Attribute-Based Architecture Styles. In *Software Architecture, Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1)*, pages 225–243, San Antonio, TX, 1999.
32. P. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, November 1995.
33. B. Liskov and J. Wing. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, November 1994.
34. N. Medvidovic and D. Rosenblum. Domains of Concern in Software Architectures and Architecture Description Languages . In *Proceedings of the USENIX Conference on Domain-Specific Languages*, pages 199–212, Santa Barbara, October 1997.
35. N. Medvidovic and R.N. Taylor. A Framework for Classifying and Comparing Architecture Description Languages. In *Proceedings of the Sixth European Software Engineering Conference together with the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 60–67, Zurich, Switzerland, 1997.

36. R. Milner. A Calculus of Communicating Systems. volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.
37. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
38. D. Park. Concurrency and Automata on Infinite Sequences. In *Theoretical Computer Science: 5th GI-Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, Berlin, 1981.
39. D. Parnas. On a "Buzzword": Hierarchical Structure. In *Proceedings IFIP Congress 74*, pages 336–339. North Holland Publishing Company, 1974.
40. D.E. Perry and A.L. Wolf. Foundations for the Study of Software Architecture. *ACM SIG-SOFT*, 17(4):40–52, October 1992.
41. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
42. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., 1996.
43. C. Stirling. Model and temporal logics for processes. In F. Moller and G.M. Birtwistle, editors, *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, August 27 - September 3, 1995, Proceedings)*, volume 1043 of *Lecture Notes in Computer Science*, pages 149–237. Springer-Verlag, Berlin, 1996.
44. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.
45. J.A. Zachman. A Framework for Information Systems Architecture. *IBM Systems Journal*, 26(3):276–292, 1987.