Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 203 (2009) 37–53
www.elsevier.com/locate/entcs

Towards Component Verification in the Generic Component Framework

Julia Padberg, Hartmut Ehrig[1]

Fakultät IV – Informatik und Elektrotechnik
Technische Universität Berlin
Germany

Fernando Orejas[2]

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Barcelona, Spain

Abstract

The intention of this paper is to extend the generic component framework presented at FASE 2002 [6] to allow component verification based on export-import implications. In the generic component framework components with explicit import, export interfaces and a body specification connected by embeddings and transformations provide hierarchical composition of components with a compositional transformation semantics.
We introduce implications that relate sentences of the import stating what the component requires to sentences of the export stating what the component guarantees. The main result of this paper is that these import-export implications are compatible with the hierarchical composition as given in [6].
The second part illustrates how this abstract concept can be instantiated to Petri net systems.

Keywords: component architectures, component verification, Petri nets, temporal logic

1 Introduction

In [6] a generic component framework for system modeling was introduced for a large class of semi-formal and formal modeling techniques. According to this concept a component consists of a body, an import, and an export interface, and connections between import and body as well as export and body. We only require having suitable notions of embeddings and transformations (e.g. refinements) between specifications. This component technique is generic as it can be instantiated with different

[1] Email: {padberg,ehrig}@cs.tu-berlin.de
[2] Email:orejas@lsi.upc.edu

doi:10.1016/j.entcs.2009.03.025

specification formalisms. Moreover, the connections can be considered generic as they also allow a great variety of instantiations. The basic idea for the generic component concept stems from data type specification, precisely the algebraic module specifications [5]. It was used for various related algebraic specification techniques as e.g. in [2,11]. The transfer to process description techniques was started in [22] where modules for graph transformation systems and local action systems were investigated. In [16] Petri net modules were introduced independently of the generic framework, but were shown to be compatible in [20]. In [6] algebraic high-level nets and in [17] deterministic automata were demonstrated to be instantiations.

In this paper we extend the component concept with import-export implications of components that are formulas given in an adequate logic. In the export interface the export statement is guaranteed independently of the component's environment provided the import requirement is met. Based on ideas presented at EKA 2006 [19] we present an approach to component verification that helps to guarantee specific properties. These properties are formalized in terms of a suitable logic over the basic properties of a specification. The underlying idea is that components guarantee specific export statements provided that the import assumptions are satisfied. So, components are equipped with an additional import-export implication. For the hierarchical composition of a requiring component and a providing component the export statement of the providing component has to imply the require assumptions of the requiring component's import. Then the result of the composition is a component that guarantees the original exports statements of the requiring component if the import assumptions of the providing component are met.

This paper is organized as follows. First we present in Section 2 the basic concepts and results at the abstract level of the generic component concept. In Section 3 we present the instantiation to place/transition net systems and temporal logic. We conclude with a discussion of related work and the practical impact of this approach.

2 Component Verification for Generic Components

As the approaches in [6,7,4] this work employs generic specifications, embeddings and transformations to form components. Since not all classes of embeddings and transformations are suitable for this purpose we have to state some general requirements first. In the concrete specification technique the validity of these requirements needs to be proved when instantiating the generic concept.

2.1 General Assumptions of the Transformation based Approach

Our generic technique requires a defined class of specifications together with transformations and embeddings. The transformations define a class of refinements for the specifications, so they are used for the connection between export interface and the component body. Since there exist so many notions of refinement, even for a single specification technique, this assumption should not be further formalized at the abstract level. Nevertheless, it has to be spelled out for the instantiation of the

concept.

We require an identity of specifications and a composition operation for transformations and embeddings.

Definition 2.1 (Extension Property)
A transformation framework $\mathcal{T}$ consists of a class of transformations that includes identical transformations, is closed under composition, and satisfies the following *extension property*:
For each transformation $trafo : SPEC_1 \Longrightarrow SPEC_2$ and each embedding $i_1 : SPEC_1 \to SPEC_1'$ there is a selected transformation $trafo' : SPEC_1' \Longrightarrow SPEC_2'$ with embedding $i_2 : SPEC_2 \to SPEC_2'$, called the *extension* of $trafo$ with respect to i_1, leading to the adjacent *extension diagram* (**1**). Intuitively, each refinement from $SPEC_1$ to $SPEC_2$ via $trafo$ can be extended to a refinement from $SPEC_1'$ to $SPEC_2'$ via $trafo'$.

$$\begin{array}{ccc} SPEC_1 & \xrightarrow{i_1} & SPEC_1' \\ {\scriptstyle trafo}\Downarrow & (\mathbf{1}) & \Downarrow{\scriptstyle trafo'} \\ SPEC_2 & \xrightarrow{i_2} & SPEC_2' \end{array}$$

Moreover, we need the possibility to compose vertically and to decompose horizontally extension diagrams as stated subsequently.

Definition 2.2 (Vertical composition of extension diagrams)
Given the diagram below and let the squares (**1**) and (**2**) be extension diagrams, then the composed square (**1** + **2**) is an extension diagram as well.

$$\begin{array}{ccc} SPEC_1 & \xrightarrow{i_1} & SPEC_1' \\ \| & & \| \\ {\scriptstyle trafo_1}\Downarrow & (\mathbf{1}) & \Downarrow{\scriptstyle trafo_1'} \\ SPEC_2 & \xrightarrow{i_2} & SPEC_2' \\ \| & & \| \\ {\scriptstyle trafo_2}\Downarrow & (\mathbf{2}) & \Downarrow{\scriptstyle trafo_2'} \\ SPEC_3 & \xrightarrow{i_3} & SPEC_3' \end{array}$$

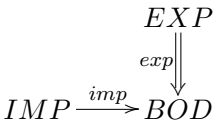
Definition 2.3 (Horizontal decomposition of extension diagrams)
Given the diagram below and let the outer square (**1** + **2**) be an extension diagram with $i_1 = i_1'' \circ i_1'$, then there exists $trafo'' : SPEC_1'' \Longrightarrow SPEC_2''$ yielding the two extension diagrams (**1**) and (**2**) below.

$$\begin{array}{ccccc} & & i_1 & & \\ & \overset{\frown}{} & & \overset{\frown}{} & \\ SPEC_1 & \xrightarrow{i_1'} & SPEC_1'' & \xrightarrow{i_1''} & SPEC_1' \\ {\scriptstyle trafo}\Downarrow & (\mathbf{1}) & \| & (\mathbf{2}) & \Downarrow{\scriptstyle trafo'} \\ & & {\scriptstyle trafo''}\Downarrow & & \\ SPEC_2 & \xrightarrow[i_2']{} & SPEC_2'' & \xrightarrow[i_2'']{} & SPEC_2' \\ & \underset{\smile}{} & & \underset{\smile}{} & \\ & & i_2 & & \end{array}$$

2.2 Components and Composition

Based on the requirements explained above, we are now able to define component specifications and the corresponding hierarchical composition operation.

Definition 2.4 (Component)
A component specification $Comp = (IMP, EXP, BOD, imp, exp)$ consists of a body specification BOD, an import specification IMP with an embedding $IMP \xrightarrow{imp} BOD$ and an export specification EXP with a transformation $EXP \overset{exp}{\Longrightarrow} BOD$.



2.3 Import-Export Implications

Components are self-contained units with a well-defined syntax and semantics. In [6] semantics of components are defined by considering each possible environment expressed by each possible transformation of the component's import. According to the transformation-based semantics the notion of import-export implications characterize the component with respect to its environment. Based on an adequate logic calculus that allows the formulation of formulas and their translation along transformations, import-export implications can be defined for components.

To define a logic over a specification we need to relate the vocabulary of the logic to the specification $SPEC$, so we need some signature Σ for $SPEC$. Then $SPEC \in \mathbf{\Sigma}$ the set of all specifications with signature Σ.

Definition 2.5 (Underlying logic)
The underlying logic $(Sen(\Sigma), \models)$ over the signature Σ consists of the set of formulas over that signature $Sen(\Sigma)$ and a relation $\models_\Sigma \subseteq \mathbf{\Sigma} \times Sen(\Sigma)$ where $\mathbf{\Sigma}$ denotes the set of all specifications with signature Σ.

Definition 2.6 (Translation of the underlying logic)
Given the underlying logic $(Sen(\Sigma), \models)$ then for each transformation $trafo : SPEC_1 \Longrightarrow SPEC_2$ there has to be a translation of sentences $\mathfrak{T}_{trafo} : Sen(\Sigma_1) \rightarrow Sen(\Sigma_2)$ with $SPEC_i \in \mathbf{\Sigma_i}$ for $1 \leq i \leq 2$.
The translation has to be compatible with the morphism composition, i.e. for transformations $trafo_i : SPEC_i \Longrightarrow SPEC_{i+1}$ with $i \leq i \leq 2$ there is the translation $\mathfrak{T}_{trafo_1 \circ trafo_2} = \mathfrak{T}_{trafo_1} \circ \mathfrak{T}_{trafo_2} : Sen(\Sigma_1) \rightarrow Sen(\Sigma_3)$ for $SPEC_i \in \mathbf{\Sigma_i}$.
The translation along an identity has to yield an identical translation, i.e.$\mathfrak{T}_{id} = \mathfrak{ID}$.

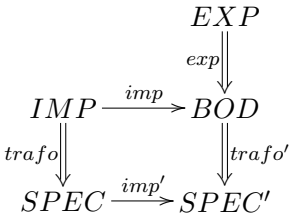
Note that $SPEC \models \varphi$ then $SPEC' \models \mathfrak{T}_{trafo}(\varphi)$ is not demanded as it is too strong for most process specification. E.g. liveness considered as a temporal logic formula over some process specification is usually not preserved by morphisms.

Definition 2.7 (Import-export implication for components)
Given a component $Comp = (IMP, EXP, BOD, imp, exp)$ then an import-export implication $\rho \Rightarrow \gamma$ consists of $\rho \in Sen(IMP)$ and $\gamma \in Sen(EXP)$.

The import-export implication provides information on the component's body at its interfaces. This information concerns the assumptions and guarantees of

a component in an arbitrary environment. So, satisfaction of the import-export implications is formulated with respect to an arbitrary environment, formalized by an arbitrary transformation of the import interface. Then we require that if this environment satisfies the translated import assumption, then the corresponding extension will satisfy the translated export statement.

Definition 2.8 (Satisfaction of an import-export implication)
Given a component $Comp = (IMP, EXP, BOD, imp, exp)$ then the import-export implication $\rho \Rightarrow \gamma$ with $\rho \in Sen(IMP)$ and $\gamma \in Sen(EXP)$ is satisfied if we have $SPEC \models \mathfrak{T}_{trafo}(\rho) \Rightarrow SPEC' \models \mathfrak{T}_{trafo' \circ exp}(\gamma)$ for all extension diagrams:



A component with guarantees is a component that ensures the export statement for any possible environment provided the import assumptions are met.

Definition 2.9 (Component with guarantees)
A component with guarantees $Comp = (IMP, EXP, BOD, imp, exp, \rho, \gamma)$ consists of a component $(IMP, EXP, BOD, imp, exp)$ together with the import-export implication $\rho \Rightarrow \gamma$ that has to be satisfied.

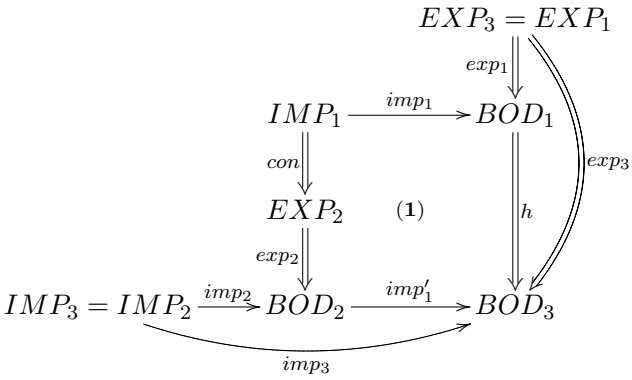
Then hierarchical composition allows the propagation of the export statements, provided the export statement of the imported component implies the import requirement of the importing component. This is defined by the connecting condition.

Definition 2.10 (Connecting condition)
Given components $Comp_i = (IMP_i, EXP_i, BOD_i, imp_i, exp_i, \gamma_i, \rho_i)$ for $i \in \{1, 2\}$ and a connection transformation $con : IMP_1 \Longrightarrow EXP_2$ then the connecting condition is satisfied if we have for all transformations $trafo : EXP_2 \Longrightarrow SPEC$:

$$SPEC \models \mathfrak{T}_{trafo}(\gamma_2) \Rightarrow SPEC \models \mathfrak{T}_{trafo \circ con}(\rho_1)$$

Definition 2.11 (Hierarchical Composition)
Given components $Comp_i = (IMP_i, EXP_i, BOD_i, imp_i, exp_i, \gamma_i, \rho_i)$ for $i \in \{1, 2\}$ and a connection transformation $con : IMP_1 \Longrightarrow EXP_2$ then the hierarchical composition $Comp_3$ of $Comp_1$ and $Comp_2$ via $con : IMP_1 \Longrightarrow EXP_2$ is defined by $Comp_3 := Comp_1 \circ_{con} Comp_2 = (IMP_3, EXP_3, BOD_3, imp_3, exp_3, \gamma_1, \rho_2)$ with $imp_3 := imp'_1 \circ imp_2$ and $exp_3 := h \circ exp_1$ as depicted below where (**1**) is an extension

diagram :

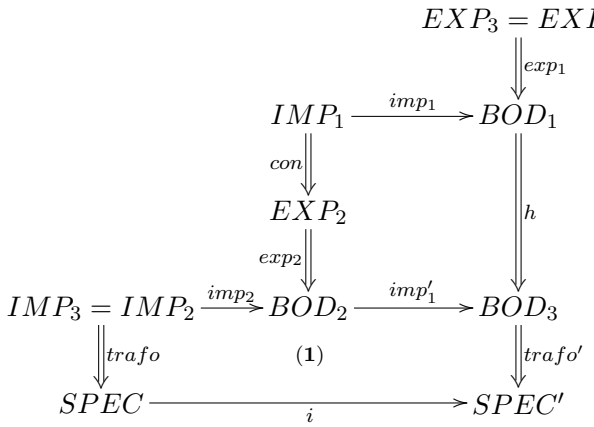


In order to have a compositional approach to component verification we now need to ensure that the hierarchical composition preserves the components guarantees in a suitable way.

Fact 2.12 ***(Hierarchical composition propagates guarantees)*** *Given components $Comp_1$ and $Comp_2$ with guarantees and a connection con : $IMP_1 \Longrightarrow EXP_2$ satisfying the connecting condition, then the result of the hierarchical composition $Comp_3 = Comp_1 \circ_{con} Comp_2$ is again a component with guarantees.*

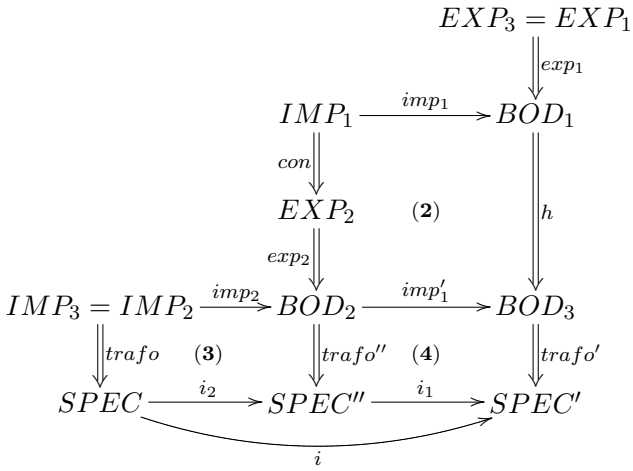
So, we have to show that the hierarchical composition $Comp_3 = Comp_1 \circ_{con} Comp_2$ satisfies the import-export implication $\rho_2 \Rightarrow \gamma_1$.

Proof. We need to show that $SPEC \models \mathfrak{T}_{trafo}(\rho_2) \Rightarrow SPEC' \models \mathfrak{T}_{trafo' \circ exp_3}(\gamma_1)$ for any extension (**1**) in the diagram below:



Due to extension decomposition (Def. 2.3) we have the three extension diagrams

(**2**), (**3**) and (**4**) with $i = i_1 \circ i_2$:



So, we have:

$SPEC \models \mathfrak{T}_{trafo}(\rho_2) \Rightarrow SPEC'' \models \mathfrak{T}_{trafo'' \circ exp_2}(\gamma_2)$
as $Comp_2$ has guarantees,

$SPEC'' \models \mathfrak{T}_{trafo'' \circ exp_2}(\gamma_2) \Rightarrow SPEC'' \models \mathfrak{T}_{trafo'' \circ exp_2 \circ con}(\rho_1)$
due to the connecting condition, and

$SPEC'' \models \mathfrak{T}_{trafo'' \circ exp_2 \circ con}(\rho_1) \Rightarrow SPEC' \models \mathfrak{T}_{trafo' \circ h \circ exp_1}(\gamma_1)$
as $Comp_1$ has guarantees and due to vertical composition in Def. 2.2

So, we directly conclude:

$$SPEC \models \mathfrak{T}_{trafo}(\rho_2) \Rightarrow SPEC' \models \mathfrak{T}_{trafo' \circ h' \circ exp_1}(\gamma_1)$$

□

3 Basic Concepts for Verification using Petri Net Components

In this section we give an instantiation of the generic framework and illustrate the basic concepts in terms of place/transition systems and temporal logic. To obtain the results of the previous section we have to ensure that the modeling technique has specific properties, namely the extension property, the composition and decomposition of extension diagrams. And the underlying logic, in this instantiation a linear time logic, needs to be provided with a suitable translation of formulas along the morphisms.

In [16] Petri net components have been first introduced. The import interface specifies resources which are used in the construction of the body, while the export interface specifies the functionality available from the Petri net component to the outside world. The body implements the functionality specified in the export interface using the imported functionality. Here, we need to treat the markings explicitly

as we want to verify behavior properties of the components.

3.1 Components of Petri Net Systems

First we give a short intuition of the underlying formalism. We use the algebraic notion of Petri nets as introduced in [12]. Hence, a place/transition (PT) net is given by the set of transitions and the set of places and the pre and post domain function; $N = (T \overset{pre}{\underset{post}{\rightrightarrows}} P^{\oplus})$ where $P^{\oplus}$ is the free commutative monoid over P– or the set of finite multisets over P. So, an element $w \in P^{\oplus}$ can be presented as a linear sum $w = \Sigma_{p \in P} \lambda_p p$ and we can extend the usual operations and relations as $\oplus$, $\ominus$, $\leq$, and so on. The initial marking (and markings in general) can be understood both as a linear sum, i.e. $\widehat{m} \in P^{\oplus}$ as well as a finitely based mapping, i.e. $\widehat{m} : P \to \mathbb{N}$.

We use much simpler morphisms than in [16] that do not preserve any specific properties as safety or liveness. The import morphism *imp* is a *plain, injective morphism* and describes where the resources of the import are used in the body. The export morphism *exp* is a *t-partial, injective morphism*. So, we have a very loose interpretation of refinement: those transition that are not mapped represent some not explicitly specified subnet of the target net.

Definition 3.1 (PT net systems and morphisms)
A PT net system $PS = (N, \widehat{m})$ is given by a PT net $N = (P, T, pre, post)$ where $pre, post : T \to P^{\oplus}$ represent the pre and post domain of a transition, and $\widehat{m} : P \to \mathbb{N}$ is the initial marking.

- A t-partial morphisms $h : PS_1 \to PS_2$ is a mapping where $h_P : P_1 \to P_2$ is a total function and $h_T : T_1 \to T_2$ is a partial function such that h is arc preserving; for all $t \in dom(f_T)$ we have: $h_P^{\oplus} \circ pre_1 = pre_2 \circ h_T(t)$ and $h_P^{\oplus} \circ post_1 = post_2 \circ f_T(t)$.
- Morphisms are plain if $h_T : T_1 \to T_2$ is a total function as well. The class of injective plain morphisms is denoted by $\mathcal{I}$.
- Morphisms are marking strict if $\widehat{m_1}(p) = \widehat{m_2}(h(p))$ for all $p \in P_1$. The class of marking strict t-partial, injective morphisms is denoted by $\mathcal{E}$.
- PT net systems and t-partial morphism comprise the category $\mathbf{PS_{tp}}$.

Note that the initial marking does not play a role in this category as the morphisms do not take it into account. So, all PT systems that consist of the same net, but have different initial markings are isomorphic in $\mathbf{PS_{tp}}$. However, we have a unique marking for extension diagrams in Fact 3.4.

The classes $\mathcal{I}$ and $\mathcal{E}$ are closed under composition and both include identities, but note that $\mathcal{I}$ is closed under composition with isomorphisms, whereas $\mathcal{E}$ is not.

Fact 3.2 ***(Pushouts in the category*** $\mathbf{PS_{tp}}$***)***
Given PT systems PS_i *for* $0 \leq i \leq 2$ *and morphisms* $PS_1 \xleftarrow{h_1} PS_0 \xrightarrow{g_1} PS_2$ *in the category* $\mathbf{PS_{tp}}$ *then here is a pushout* $PS_1 \xrightarrow{h_2} PS_3 \xleftarrow{g_2} PS_2$ *that can be constructed component-wise for places and transitions and with an arbitrary initial marking for* PS_3.

The proof (for details see [18]) uses the standard construction of pushouts in the category of sets **Set** and in the category of partial sets **parSet**.

Petri net components consist of three PT systems: the import PT system $(IMP, \widehat{m_I})$, the export PT system $(EXP, \widehat{m_E})$, and the body PT system $(BOD, \widehat{m_B})$. Note that there is no marking compatibility required for $imp \in \mathcal{I}$. This allows the deletion of parts of the initial marking during the hierarchical composition, and is needed to remove "pseudo-initial" tokens that will be provided by the environment (see Subsect. 3.3).

Definition 3.3 (PT system component)
A PT system component $PC = ((IMP, \widehat{m_I}), (EXP, \widehat{m_E}), (BOD, \widehat{m_B}), imp, exp)$ consists of the import PT system $(IMP, \widehat{m_I})$, the export PT system $(EXP, \widehat{m_E})$, the body PT system $(BOD, \widehat{m_B})$, and of two morphisms $(IMP, \widehat{m_I}) \xrightarrow{imp \in \mathcal{I}} BOD, \widehat{m_B}) \xleftarrow{exp \in \mathcal{E}} (EXP, \widehat{m_E})$.

Extension diagrams are pushouts where one morphism is marking strict and injective and the other is injective and plain. We need to prove that these can be constructed component-wise for any pair of $\mathcal{I}$- and $\mathcal{E}$-morphisms.

Fact 3.4 ***(Extension diagrams of*** $\mathcal{I}$***- and*** $\mathcal{E}$***-morphisms)***
Given PT nets $N_i = (P_i, T_i, pre_i, post_i)$ *and PT systems* $PS_i = (N_i, \widehat{m_i})$ *for* $0 \leq i \leq 2$ *and the morphisms* $PS_1 \xleftarrow{h_1} PS_0 \xrightarrow{g_1} PS_2$ *where* $h_1 \in \mathcal{I}$ *and* $g_1 \in \mathcal{E}$ *then there is a pushout* $PS_1 \xrightarrow{h_2} PS_3 \xleftarrow{g_2} PS_2$ *with*

$$\widehat{m_3}(p) = \begin{cases} \widehat{m_2}(p_2) \ ; g_2(p_2) = p \notin h_2(P_1) \\ \widehat{m_1}(p_1) \ ; h_2(p_1) = p \end{cases}$$

that is an extension diagram with $g_2 \in \mathcal{I}$ *and* $h_2 \in \mathcal{E}$.

Proof.

$PS_3 = (N_3, \widehat{m_3})$ is pushout by construction.

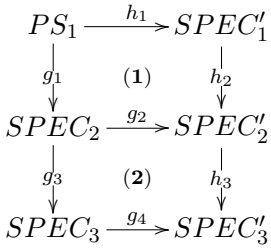
$$\begin{array}{ccc} PS_0 & \xrightarrow{h_1} & PS_1 \\ {\scriptstyle g_1}\downarrow & (1) & \downarrow{\scriptstyle h_2} \\ PS_2 & \xrightarrow{g_2} & PS_3 \end{array}$$

Plain morphisms are preserved as total morphisms are pushout stable in **parSet**. Injective morphisms are preserved as injective morphisms are pushout stable in **Set** as well as in **parSet**. The construction of $\widehat{m_3}$ directly yields that h_2 is marking strict. So, $g_2 \in \mathcal{I}$ and $h_2 \in \mathcal{E}$. Moreover, the initial marking $\widehat{m_3}$ is uniquely determined by the requirement $h_2 \in \mathcal{E}$. □

Remark 3.5 (Pushout construction)
It is interesting to note that the construction of the PT net N_3 coincides with the corresponding construction in [20] where we have used substitution morphisms instead of t-partial morphisms. In fact, the transitions T_3 of N_3 can be constructed as $T_3 = (T_1 - h_{1T}(T_0)) + T_2$ with inclusion g_{2T} and partial function h_{2T}. These are jointly surjective, s.t. $pre_3(t)$ – and similar $post_3$ – is uniquely defined by $pre_2(t)$ for $t \in T_2$ and by $pre_1(t)$ otherwise (for details see [18]).

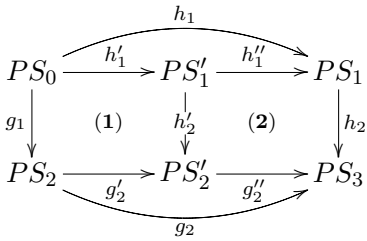
Fact 3.4 yields the extension property in Def. 2.1. Together with the composition and decomposition of pushouts it also yields the vertical composition in Def. 2.2 and horizontal decomposition in Def. 2.3 of extension diagrams.

Fact 3.6 ***(Vertical composition of extension diagrams)***
Given the diagram below and let the squares $(\mathbf{1})$ *and* $(\mathbf{2})$ *be extension diagrams, then the composed square* $(\mathbf{1}+\mathbf{2})$ *is an extension diagram as well.*



Proof. The composition of pushouts yields the pushout $(\mathbf{1}+\mathbf{2})$. Since marking-strict, injective, and t-partial morphisms are closed under composition respectively, we have $g_3 \circ g1 \in \mathcal{E}$ as well as $h_3 \circ h_2 \in \mathcal{E}$ and $h_1, g_4 \in \mathcal{I}$ by assumption. Hence, $(\mathbf{1}+\mathbf{2})$ is an extension diagram. □

Fact 3.7 ***(Horizontal decomposition of extension diagrams)***
Given the diagram below and let the outer square $(\mathbf{1}+\mathbf{2})$ *be an extension diagram with* $h_1 = h_1'' \circ h_1'$*, then there exists* $h_2' : PS_1' \Longrightarrow PS_2'$ *yielding the two extension diagrams* $(\mathbf{1})$ *and* $(\mathbf{2})$ *below.*



Proof. For the decomposition we construct $(\mathbf{1})$ as an extension diagram, so it is pushout and $h_2' \in \mathcal{E}$. As $(\mathbf{1}+\mathbf{2})$ and $(\mathbf{1})$ are pushouts, and $(\mathbf{2})$ with g_2'' as the induced pushout morphism commutes, we use the pushout decomposition property to conclude that $(\mathbf{2})$ is pushout as well. As plain, injective morphisms are pushout stable we have that $(\mathbf{1})$ and $(\mathbf{2})$ are extension diagrams. □

3.2 Temporal Logic

We use a notation closely related to standard linear time logics (LTL) as e.g. in [13] or [9]. For each net we assume a set of atomic propositions AP over the markings of the net. For a marking $m \in P^{\oplus}$ the satisfaction of a atomic proposition is given if the proposition $\mathbf{p}$ is true for m.

A LTL formula is an element of the language

$$\mathbf{f} := \mathbf{p} \,|\, \neg \mathbf{f} \,|\, \mathbf{f} \wedge \mathbf{f} \,|\, \mathbf{X}\, \mathbf{f} \,|\, \mathbf{f}\, \mathbf{U}\, \mathbf{f}$$

constructed out of atomic propositions $\mathbf{p}$ to which boolean connections $\neg$ (negation) and $\wedge$ (conjunction), as well as the temporal operators "until" $\mathbf{U}$ and "next" $\mathbf{X}$ are applied.

Since a LTL requires runs of a system we now define runs of a PT system $(N, \widehat{m})$ as an infinite sequence of markings $\delta := m_0 \cdot m_1 \cdot m_2 \cdot ...$ where $m_0 = \widehat{m}$ is the initial marking. Either we have some $t \in T$ for each $i \geq 0$ so that $m_i[t > m_{i+1}$ or we repeat the last marking, i.e. if there is no $t \in T$ such that $m_i[t > m_{i+1}$ then $m_j = m_i$ for all $j > i$.

We assume a set of atomic propositions AP on markings, so that for each marking $\pi : P^{\oplus} \to 2^{AP}$ assigns truth values to the propositions. Thereby we have $\pi(m)(\mathbf{p}) = true$ for $\mathbf{p} \in AP$ and $m \in P^{\oplus}$ is denoted by $\mathbf{p} \in \pi(m)$.

Then we define inductively for formulas $\mathbf{f}$:

- for an atomic proposition $(\delta, j) \models \mathbf{p}$ iff $\mathbf{p} \in \pi(m_j)$ for $\mathbf{p} \in AP$
- for the boolean operators $(\delta, j) \models \neg \mathbf{f} \in AP$ iff not $(\delta, j) \models \mathbf{f}$
  $(\delta, j) \models \mathbf{f_1} \wedge \mathbf{f_2} \in AP$ iff $(\delta, j) \models \mathbf{f_1}$ and $(\delta, j) \models \mathbf{f_2}$
- for the until operator $(\delta, j) \models \mathbf{f_1}\, \mathbf{U}\, \mathbf{f_2}$ iff
  there is some $k \geq j$ with $(\delta, k) \models \mathbf{f_2}$
  and for all $j \leq i \leq k$ holds $(\delta, i) \models \mathbf{f_1}$
- for the next operator $(\delta, j) \models \mathbf{X}\, \mathbf{f}$ iff $(\delta, j+1) \models \mathbf{f}$

We abbreviate formulas using the usual boolean operators as they can be defined using the negation and the conjunction. Analogously we can define further temporal operators as "eventually" or "future" $\mathbf{F}$ by $\mathbf{F}\, \mathbf{f} := true\, \mathbf{U}\, \mathbf{f}$ and the operator "always" or "globally" $\mathbf{G}\, \mathbf{f} := \neg\, \mathbf{F}\, \neg\, \mathbf{f}$. The set of all LTL formulas with respect to the set of atomic propositions AP is denoted by $\mathbb{F}$.

A net system $(N, \widehat{m}) \models \mathbf{f}$ satisfies an LTL formula $\mathbf{f} \in \mathbb{F}$ if for all runs δ of $(N, \widehat{m})$ we have $(\delta, 0) \models \mathbf{f}$.

Definition 3.8 (Underlying logic for PT system components)
The underlying temporal logic $(\mathbb{F}, \models)$ over the net N consist of the formulas $\mathbb{F}$ over the net N and the relation $\models_N \subseteq \mathbf{N} \times \mathbb{F}$ where $\mathbf{N} = \{(N, \widehat{m}) | \widehat{m} \in P^{\oplus}\}$ the set of all PT systems consisting of the net N and some initial marking $\widehat{m} \in P^{\oplus}$.

Next we define the translation of LTL formulas based on a mapping of the atomic

propositions that is compatible with the mapping of the places and show then to be compatible with the composition of morphisms as required in Def. 2.6.

Definition 3.9 (Translation of a formula)
Given PT systems $(N_i, \widehat{m_i})$ with atomic propositions AP_i and $\pi_i : P_i^{\oplus} \to AP_i$ for $1 \leq i \leq 2$, a morphism $h : PS_1 \to PS_2$, and a mapping of the atomic propositions $h_{AP} : AP_1 \to AP_2$ that is compatible with the mapping of the places, i.e. $\pi_2 \circ h_P^{\oplus} = h_{AP} \circ \pi_1$, then we define $\mathfrak{T}_h : \mathbb{F}_{AP_1} \to \mathbb{F}_{AP_2}$ inductively:

- for atomic propositions $\mathfrak{T}_h(\mathbf{p}) := h_{AP}(\mathbf{p})$
- for the boolean operators $\mathfrak{T}_h(\neg \mathbf{f}) := \neg \mathfrak{T}_h(\mathbf{f})$
  $\mathfrak{T}_h(\mathbf{f_1} \wedge \mathbf{f_2}) := \mathfrak{T}_h(\mathbf{f_1}) \wedge \mathfrak{T}_h(\mathbf{f_2})$
- for the until operator $\mathfrak{T}_h(\mathbf{f_1}\ \mathbf{U}\ \mathbf{f_2}) := \mathfrak{T}_h(\mathbf{f_1})\, \mathbf{U}\, \mathfrak{T}_h(\mathbf{f_2})$
- for the next operator $\mathfrak{T}_h(\mathbf{X}\, \mathbf{f}) := \mathbf{X}\, \mathfrak{T}_h(\mathbf{f})$
- for the eventually operator $\mathfrak{T}_h(\mathbf{F}\, \mathbf{f}) := \mathbf{F}\, \mathfrak{T}_h(\mathbf{f})$
- for the always operator $\mathfrak{T}_h(\mathbf{G}\, \mathbf{f}) := \mathbf{G}\, \mathfrak{T}_h(\mathbf{f})$

Fact 3.10 ***(Composition of Translation)***
Given mappings of atomic propositions $h_{AP} : AP_1 \to AP_2$ and $g_{AP} : AP_2 \to AP_3$ compatible with $h : PS_1 \to PS_2$ and $g : PS_2 \to PS_3$, then we have $\mathfrak{T}_g \circ \mathfrak{T}_h = \mathfrak{T}_{g \circ h}$.

Proof. We have compatibility of $g \circ h : PS_1 \to PS3$ with π_1 and π_3 due to $\pi_3 \circ (g_P \circ h_P)^{\oplus} = g_{AP} \circ \pi_2 \circ h_P^{\oplus} = g_{AP} \circ h_{AP} \circ \pi_1$. Moreover $(g \circ h)_{AP} = g_{AP} \circ h_{AP}$, so we can prove inductively the composition of the translations. □

Results 3.11 (Component-based Verification) We now have

- PT system components with guarantees (see Def. 2.9), and
- hierarchical composition propagating guarantees (see Def. 2.12).

Due to the fact that PT systems are an instantiation of the generic component framework in Sect. 2.

3.3 Example

In this section we give an example to illustrate our approach. The example is merely a structural example without a specific meaning. For the practical impact of this approach see the discussion in Section 4 or [21].

In our example the set AP of of atomic propositions on markings of a PT system $(N, \widehat{m})$ with places P is given by $AP = \mathbb{N} \times P$ and for each marking $m : P \to \mathbb{N}$ we have $\pi(m) = \{(m(p), p) | p \in P\}$. This means $(n, p) \in AP$ is true under marking m if $n = m(p)$. This allows the definition of the mapping $h_{AP} : AP_1 \to AP_2$ with $h_{AP}(n, p) = (n, h_P(p))$. This mapping is well-defined as translations are given for $\mathcal{E}$-morphisms only, so h_P is injective.

In Fig. 2 we consider a component $Comp_1 = (IMP_1, EXP_1, BOD_1, \rho_1, \gamma_1)$ where the export statement $\gamma_1 := (\mathbf{G}\, \mathbf{F}\, p5)$ ensures that the marking with one

token on place $p5$ is always reachable in any extension of BOD_1, provided that the corresponding extension of the import behaves like the transition $t2$, i.e. once there is a token on the pre-place eventually there will be a token on the post-place and it will stay there. This is denoted by the import requirement $\rho_1 := (p3 \Rightarrow \mathbf{F}\,\mathbf{G}\,p4)$. This excludes refinements of the import where for example a transition removes the token from place $p4$, or where a transition puts more than one token to place $p4$.

In the subsequent figures morphisms are indicated by identical names of places and transitions. Those nodes that are not in the codomain of a morphism remain without a name. So, the morphisms are all inclusions and the translations of formulas are identities and hence omitted in this example.

Fig. 3 depicts the component $Comp_2 = (IMP_2, EXP_2, BOD_2, \rho_2, \gamma_2)$ where $\rho_2 := (p1 \Rightarrow \mathbf{F}\,\mathbf{G}\,p2)$ expresses that if there is one token on place $p1$ then eventually there will be one token on place $p2$ and it will stay there. The export statement $\gamma_2 := (p3 \Rightarrow \mathbf{F}\,\mathbf{G}\,p4)$ states the same for the places $p3$ and $p4$.

Both components $Comp_1$ and $Comp_2$ are components with guarantees, that is the import-export implications hold for arbitrary extensions. The body of $Comp_1$ still satisfies γ_1 if the import is extended by an arbitrary PT system PTS in such a way that if a token is on place $p3$ then there will be a token on place $p4$ and it will stay there. This is quite obvious and the argument is given here informally: by construction of the extension diagram the PT system PTS will be glued into the PT system BOD_1 as sketched in Fig. 1 and this PT system will still satisfy the temporal logic formula $\gamma_1 := (\mathbf{G}\,\mathbf{F}\,p5)$ provided the PT system PTS satisfies $\rho_1 := (p3 \Rightarrow \mathbf{F}\,\mathbf{G}\,p4)$. The argument that component $Comp_2$ satisfies its import-export implication is similar.

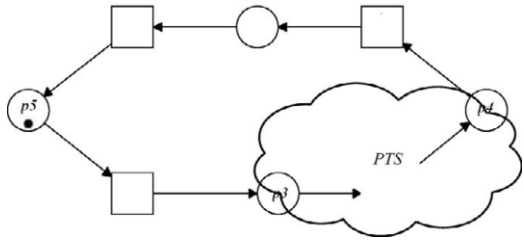


Fig. 1. Arbitrary extension of BOD_1

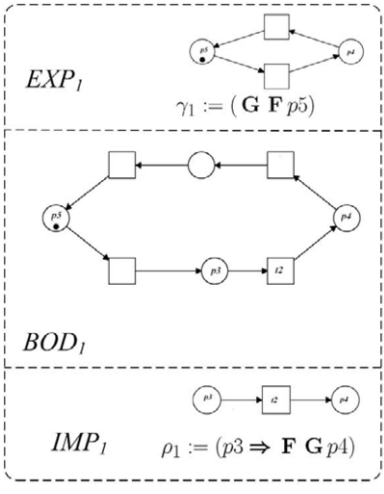


Fig. 2. $Comp_1$

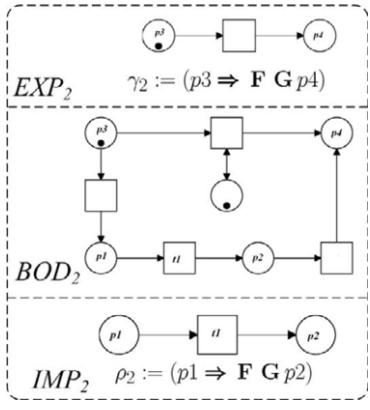


Fig. 3. $Comp_2$

The composition of the two components $Comp_1 \circ_{con} Comp_2 = Comp_3 = (IMP_2, EXP_1, BOD_3, \rho_2, \gamma_1)$ is depicted in Fig. 4. It is achieved by gluing the bodies BOD_1 and BOD_2 along IMP_1 resulting in BOD_3. The connecting transformation maps the system $(IMP_1, \widehat{m_{I1}})$ to the system $(EXP_2, \widehat{m_{E2}})$ regardless of the markings.

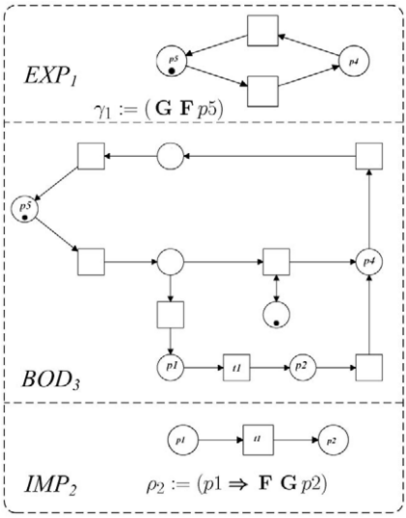


Fig. 4. $Comp_3$

The component $Comp_3$ now guarantees that a marking with a token on place $p5$ can be always reached again, provided the extension of the import satisfies the import requirement that a token on place $p1$ implies that there is eventually a token on place $p2$ and it will stay there. Note that a token in the initial marking of the body needs not be represented in the export as well. In that case it is independent of the environment and has to be preserved. The marked place in the middle of BOD_2 in component $Comp_2$ (in Fig. 3) is preserved by the composition and is still marked in BOD_3 in component $Comp_3$. If a token in the initial marking of the body is represented in the export as well, then it may be dependent on the importing component, and hence needs to be deleted by the composition. So, a token in the initial marking of the export can be either one that is required by the component importing it or it is provided by the firing of the importing component. This is for the imported component indistinguishable. Either the token is represented in the initial marking of the import of the importing component then it is preserved, or it is not represented in the import then it is deleted. So, "pseudo-initial" tokens are eliminated by the construction of the extension diagram. The token on place $p3$ in BOD_2 and EXP_2 is an example therefore.

3.4 Open Questions

As this paper is a first step towards component verification in the generic component framework open questions are still left, e.g.:

- *Translation of formulas*
  In this approach here we use a translation that is defined place-wise. It is a straightforward approach that it easy to follow. But it has the drawback that the initial markings are not necessarily mapped onto each other. So, for example the formula $\gamma_1 := (\mathbf{G}\ \mathbf{F}\ p5)$ states that the system $(EXP_1, \widehat{m_{E1}})$ is reversible. But for the system $(BOD_3, \widehat{m_{B3}})$ this formula does not state reversibility.

- *Quantifiers*
  In order to translate formulas with quantifiers the scope of the formula needs to be extended according to the target system. Then it would be possible to translate formulas that deal with all reachable markings or all transitions, as needed for example for liveness.
- *Satisfaction of import-export implications*
  Since the import-export implications have to hold for arbitrary extensions the techniques for model checking cannot be applied. In the examples this has not yet been a problem. But for larger applications some proof technique would be very desirable.

4 Conclusion

Summarizing, we have an assume-guarantee approach to component-based verification that is independent of the underlying specification technique. Formally, a component is given by three specifications, the body specification, the import and the export interface. To express properties of components, an appropriate logic formalism has to be required that allows expressing the desired properties. A component is then equipped with two additional logic formulas that represent the import-export implication. The import assumptions describe in an abstract way the properties the underlying component needs to have to ensure the desired behavior. Then the export guarantees some property denoted by the export statement. Hierarchical composition allows concluding the import-export implication where the providing component's import assumption implies the requiring component's export statement.

4.1 Related work

In model checking the typical approach to verification of components is to check the properties for all possible environments. But there are various approaches, e.g. [10,3,8] that share the underlying hypothesis that the required property can be achieved only in specific environments. In [8] a framework for assume-guarantee paradigm is introduced that uses labeled transition systems to model the behavior of communicating components in a concurrent system. In [3] the interfaces are modeled using input/output automata. The parallel composition of the interfaces is given and criteria for the compatibility are presented, but this approach merely concerns the interfaces. In [10] certain properties, as deadlock freedom are checked based on assumptions that the component makes about the expected interaction behavior of other components.

In [1] concurrent automata are introduced that describe the concurrent behavior of input and output ports in terms of their operations. Considering the automata as the components body and the input ans output ports as the import and export interfaces, respectively, maybe allows fitting this approach into the general framework presented in this paper.

4.2 Practical Impact

The area of controls for discrete event based systems needs an approach of modeling and structuring systems as well as the verification of the systems properties. In [21] we propose to model and verify system properties of discrete event based systems using Petri nets components. Based on import-export implications of Petri net components the temporal logic formula given in the export interface is guaranteed independently of the component's environment. We investigate the approach's feasibility for controlling a technical system and describe parts of a model plant for a packing process using Petri net components. The verification of basic properties makes use of the hierarchical composition and the propagation of the import-export implications.

The Petri net based sequence controller is modeled using the tool Netlab [14] which is a modeling, analysis and simulation environment that also supports the design and synthesis of discrete event - or hybrid systems under Matlab/Simulink. Netlab is a graphic P/T net editor, that allows loading and saving in PNML [15]. We intend to add structuring and verification means to Netlab based on Petri net components as introduced in this paper.

References

[1] J.F.K. Bowles and S. Moschoyiannis. Concurrent logic and automata combined: a semantics for components. In *Proc. of CONCUR 2006 - Foundations of Coordination Languages and Software Architectures (FOCLASA'06)*. Electronic Notes in Theoretical Computer Science, Elsevier Science, 2007. to appear.

[2] F. Cornelius, M. Baldamus, H. Ehrig, and F. Orejas. Abstract and behaviour module specifications. *Mathematical Structures in Computer Science*, 9:21–62, 1999.

[3] L. de Alfaro and T.A Henzinger. Interface automata. In *ESEC/FSE 01: Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, 2001.

[4] H. Ehrig, B. Braatz, M. Klein, F. Orejas, S. Pérez, and E. Pino. Object-oriented connector-component architectures. In *Proc. International Workshop on Formal Foundations of Embedded Software and Component-based Software Architectures (FESCA 2005)*, volume 141 of Electronic Notes in Theoretical Computer Science, Elsevier Science, pages 123–151, 2005.

[5] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, volume 21 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1990.

[6] H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A Generic Component Concept for System Modeling. In *Proc. FASE 2002: Formal Aspects of Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 32–48. Springer Verlag, 2002.

[7] H. Ehrig, J. Padberg, B. Braatz, M. Klein, F. Orejas, S. Pérez, and E. Pino. A generic framework for connector architectures based on components and transformations. In *Proc. International Workshop on Formal Foundations of Embedded Software and Component-based Software Architectures (FESCA 2004)*, volume 108 of Electronic Notes in Theoretical Computer Science, Elsevier Science, pages 53–67, 2004.

[8] D. Giannakopoulou, C. Păsăreanu, and H. Barringer. Assumption generation for software component verification. In *Proceedings of ASE-2002: The 17th IEEE Conference on Automated Software Engineering*. IEEE CS Press, 2002.

[9] C. Girault and R. Valk, editors. *Systems Engineering: a Guide to Modelling, Verification and Applications*. Springer, 2003.

[10] Paola Inverardi, Alexander L. Wolf, and Daniel Yankelevich. Static checking of system behaviors using derived component assumptions. *ACM Trans. Softw. Eng. Methodol.*, 9(3):239–272, 2000.

[11] Rosa M. Jiménez and Fernando Orejas. An algebraic framework for higher-order modules. In *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems*, volume 1709 of *Lecture Notes in Computer Science*, pages 1778–1797. Springer, 1999.

[12] J. Meseguer and U. Montanari. Petri Nets are Monoids. *Information and Computation*, 88(2):105–155, 1990.

[13] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer Verlag, 1995.

[14] Institut für Regelungstechnik, Rheinisch-Westfälische Technische Hochschule Aachen. *Petrinetz-Tool Netlab (Windows)*, 2007. http://www.irt.rwth-aachen.de/typo3/index.php?id=101&L=0.

[15] Ph. Orth, U. Küssel, and D. Abel. Netlab und Netlab Toolbox für Mat-lab/Simulink-ein Werkzeug zum Rapid Control Prototyping von Steuerungen mittels Petrinetzen. In *Tagungsband Entwurf komplexer Automatisierungssysteme EKA 2006*, pages 343–353. 2006.

[16] J. Padberg. Petri net modules. *Journal on Integrated Design and Process Technology*, 6(4):121–137, 2002.

[17] Julia Padberg. Integration of the generic component concepts for system modeling with adhesive HLR systems. *EATCS Bulletin*, 87:138–155, 2005.

[18] J. Padberg. Formal foundation for transformation-based approach to component verification. Technical Report 2006-12, Technische Universität Berlin,Fakultät IV, 2006.

[19] J. Padberg. Formale Techniken für die Beschreibung von Software-Architekturen. In R. Reussner and W. Hasselbring, editors, *Handbuch der Software-Architektur*, pages 465–476. d-punkt Verlag, 2006.

[20] J. Padberg and H. Ehrig. Petri net modules in the transformation-based component framework. *Journal of Logic and Algebraic Programming*, 67:198–225, 2005.

[21] Julia Padberg and Uwe Küssel. A component-based verification approach based on Petri net components. In *Proc. FORMS/FORMAT 2007 - "Formal Methods for Automation and Safety in Railway and Automotive Systems"*, pages 40–50. GZBV, 2007.

[22] M. Simeoni. *A Categorical Approach to Modularization of Graph Transformation Systems using Refinements*. PhD thesis, Università Roma *La Sapienza*, 1999.