Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) – Proc. GT-VMT 2001
URL: http://www.elsevier.nl/locate/entcs/volume50.html 28 pages

# A Conceptual and Formal Framework for the Integration of Data Type and Process Modeling Techniques

Hartmut Ehrig

*Technical University of Berlin, Germany*

Fernando Orejas

*Technical University of Catalunya, Spain*

**Abstract**

A conceptual framework for the integration of data type and process modeling techniques, called integration paradigm, has been presented by the authors in previous papers already. The aim of this paper is to give a short review of this conceptual framework and to present a formal model for the integration paradigm. The formal model for the four layers, called data type, data states and transformations, processes and system architecture layers respectively, is based on an integration of abstract data types and structured transition systems. This formal model can be instantiated by all kinds of basic and integrated modeling techniques. Algebraic high-level nets, attributed graph transformation, an integration of Z with statecharts, and some diagram techniques of UML are discussed on the conceptual level. As instantiation of the formal model, a well-known CCS sender specification, place/transition nets, algebraic high-level nets and attributed graph transformation are presented in this paper, while instantiations of other modeling techniques will be discussed elsewhere.

## 1 Introduction

The integration of different kinds of data type and process modeling techniques has become an important issue for the modeling of software systems in computer science and all kinds of applications in science and engineering. The data type and process view of a system are two basic views which are either modeled separately by different formalisms or jointly by an integrated formalism. We roughly distinguish the following classes of data type specification and modeling formalisms:

- algebraic/axiomatic approaches,

- state/model-oriented approaches,
- class-oriented approaches.

The main classes of process specification and modeling formalisms are the following:

- Petri net approaches,
- process algebraic approaches
- automata/statechart-oriented approaches,
- graph transformation approaches.

In each of these classes there are low level variants, where data types are only supported in a weak way by fixed data domains of alphabets, and also high level variants that are defined by integration of basic process and data type modeling techniques. Based on these examples of data type and process modeling techniques we have introduced in [EO98b,EO00] an integration paradigm for system specification and modeling on four different layers, which provides a unified approach on a conceptual level. The integration paradigm is an extension of previous approaches in our papers [EO94] and [EBC+96] motivated by [AZ95], [DG94], and [PP91].

The aim of this paper is to review the conceptual framework in [EO98b,EO00], to present a formal model of the integration paradigm, and to instantiate the formal framework by some well-known basic and integrated modeling techniques. Our formal model has been influenced by the semantical reference models for formalization and integration of Cornelius [Cor98] and Große-Rhode [Gro98,Gro99,Gro00,Gro01]. Especially it is closely related to the notion of transformation systems in [Gro98,Gro00] concerning the formal model of layers 1-3 of our integration paradigm. The formal model for layer 4 (system architecture) provides a basic version of a component concept which is extended and presented in more detail in our paper [EO01].

In section 2 of this paper we present the four layers of the integration paradigm, called data type, data state and transformation process, and system architecture layer respectively, and discuss as typical examples some well-known integrated data type and process modeling techniques. The formal model for the integration paradigm is presented in section 3. Instantiations of the formal model for CCS, low level and high level Petri nets, and attributed graph transformation are given in section 4, where for the last two instantiations we only present the main ideas. The conclusion in section 5 compares our formal model with the notion of transformation systems in [Gro98,Gro00] and an extension of the basic component concept in layer 4 to a more general concept in [EO01].

*Acknowledgements*

We are most grateful to several colleagues concerning the discussion of integrated modeling techniques, especially to Martin Große-Rhode for valuable comments concerning the draft of this paper. Thanks also to the organizers of the GT-VMT 2001 workshop for the invitation of this paper and to Maria Oswald with support from Claudia Ermel for excellent typing. This work is partially supported by the German DFG project IOSIP within the DFG priority program "Integration of Software Specification Techniques for Applications in Engineering" and by the Spanish project HEMOSS (TIC 98-0949-C02-01).

## 2 Integration Paradigm for Data Type and Process Specification and Modeling Techniques

As typical examples for the integration of data type and process specification formalisms we consider algebraic high-level nets [PER95], an integration of algebraic specification and Petri nets, attributed graph transformation [ELO95], an integration of algebraic specification and graph transformation, $\mu SZ$ [BGK98], an integration of $Z$ and statecharts, and UML [UML00] comprising different object-oriented modelling techniques.In all these and several other examples like LOTOS [Bri89], an integration of algebraic specification and CCS, there is a common pattern how the data type and the process view are combined with each other. This common pattern has been formulated as an integration paradigm in [EO98b,EO98a] and consists of four layers which are organized in a hierarchical way.

### *2.1 Integration Paradigm*

*Layers of Integration*

The first layer corresponds exactly to the data type view of the system. The following layers are integrated views of data type, data state and system architecture aspects, where each layer is based on the previous one. This hierarchical concept is different from a dimensions concept (see e.g. [Cor98,GKP98]) where dimensions are considered to be independent and can be combined in different ways.

*Layer 1: Data Types*

Layer 1 provides the description of data values and operations on data values for the system. The corresponding data type can be considered as static within the system.

*Layer 2 : Data States and Transformations*

Layer 2 provides the description of data states and data state transformations. This includes all states and transformations, which in principle can occur in the system, even if the data states are non-realizable or non-reachable by the processes of the system.

*Layer 3: Processes*

Layer 3 provides processes based on layers 1 and 2. Processes are the activities of the system according to its aims. Especially they realize the scenarios which are required from the application point of view. Moreover, there are communication mechanisms for processes with each other and with the environment in the sense of concurrent systems.

*Layer 4: System Architecture*

The system architecture should provide a modular structure of the system in terms of components, where in general each component is given by data types and a set of communicating processes as defined on layer 3. Vice versa this means that there should be horizontal structuring mechanisms in each layer 1-3 leading to a module concept in layer 4 which allows to compose components with suitable notions of compositionality.

A more detailed presentation of these layers together with several examples is given in [EO98a]. In the following we classify our three integrated formalisms mentioned above and UML according to the levels of this integration paradigm (see table 1). A more detailed discussion of algebraic high-level nets and attributed graph transformation are given in section 4 as instantiations of the formal model for the integration paradigm in section 3.

### *2.2 Algebraic High-Level Nets*

In the integrated formalism of algebraic high-level nets [PER95] the data type layer 1 is given by algebras defined by algebraic specifications such that the tokens of high-level nets are no longer black tokens, but data tokens as elements of algebras. In layer 2 the data states are given by the markings of places by data tokens and the transformation of states by firing of transitions. In layer 3 we have net processes, which can be defined for high-level nets similar to those in the low-level case. In the system architecture level 4 we have well-known parameterization and modularization concepts for algebraic specifications [EM85,EM90], net transformations defined by rule-based transformation of nets, and the concepts of union and fusion to build up larger nets from smaller components. However, a suitable modularization concept including data type and net parts is still missing.

Table 1
Integrated Data Type and Process Formalisms

| **Specification Technique** | **Layer 1 Data Types** | **Layer 2 Data States & Transformations** | **Layer 3 Processes** | **Layer 4 System Architecture** |
|---|---|---|---|---|
| **Algebraic High-Level Nets** | data tokens and algebras defined by algebraic specification | marking of places by data tokens & firing of transitions | net processes | parameterization, net transformations union / fusion |
| **Attributed Graph Transformation** | attributes and algebras defined by algebraic specification | attributed graphs and graph transformation | graph processes | parameterization, composition, modularization |
| $\mu SZ$ | type definition in $Z$ | data states and operation schemas in $Z$ | statechart processes | configurations |
| **UML** | basic data types defined by class diagrams | classes, attributes & methods | object-oriented statecharts & sequence diagrams | packages |

### *2.3 Attributed Graph Transformation*

Similar to algebraic high-level nets the specification formalism of attributed graph transformations [ELO95] is based on algebraic specifications concerning the data type layer 1. Data states in layer 2 are given by attributed graphs, which are graphs attributed by data elements of algebras defined in layer 1. Transformation of data states in layer 2 is given by the concept of graph transformation applied to attributed graphs. The notion of processes has been extended from Petri nets to graph grammars [CMR96] and in addition the concept of programmed graph transformations is useful for the concept of processes in layer 3. Concerning layer 4 we again have the well-known structuring concepts parameterization and modularization for algebraic specifications and several notions of composition and modularization for graph transformation systems are under development [EE96,HEET99].

### *2.4 $\mu SZ$*

The specification formalism $\mu SZ$ [BGK98] was developed in the ESPRESS-project for the specification of safety critical embedded systems. The main idea of $\mu SZ$ is to integrate statecharts [Har87] with Z [Spi92] and to have a notion of configurations as modular structuring mechanism. This means that in the data type layer 1 we have type definition in Z. Data states and transformations in layer 2 are given by data state and operation schemas in Z notation. The processes in layer 3 are defined by statecharts and the system architecture in layer 4 is given by configurations of $\mu SZ$.

### *2.5 UML*

The *Unified Modelling Language* UML [UML00] combines several semi-formal diagram techniques for object-oriented modelling, where a semantical foundation and integration of all these techniques is still missing but under development in different projects. We here only consider some of these diagram techniques. The data types in layer 1 are given by basic data types defined by class diagrams. Data states and transformations in layer 2 are given by classes defining the internal states by attributes and state transformations by methods. Processes in layer 3 within one object are defined using object-oriented statecharts [HG96] while sequence diagrams model processes between different objects. The package concept is a first step towards a module concept for layer 4. For class diagrams [Kla99] and object-oriented statecharts [MK98] a formal semantics is given in the general framework of metamodeling (see [GKP98]) based on Object-Z [DRS94].

## 3 A Formal Model for the Integration Paradigm

In this section we present a formal model for our integration paradigm reviewed in the previous section. The formal model is presented on a syntactical and semantical level. It is based on signatures and algebras for layers 1 and 2 as well as transition systems for layers 2 and 3 and integrated models for layer 4. In our paper [EO01] we extend this model by signature morphisms and constraints leading to an institution of integrated specification formalism and models which is the basis for a corresponding component concept.

### *3.1 Layer 1: Data Types*

Layer 1 provides the description of data values and operations on data values for the system. The corresponding data type can be considered as static within the system.The *data value signature* $SIG_0$ is an algebraic signature $SIG_0 = (S_0, \Sigma_0)$ consisting of a set $S_0$ of sorts and a family $\Sigma_0$ of operation symbols (see e.g. [EM85]) or a first order signature$SIG_0 = (S_0, \Sigma_0, \Pi_0)$, where $\Pi_0$ is a family of predicate symbols.On the semantical level we allow to have

partial algebras. However, total algebras are sufficient in several applications. In layer 1 we assume to have a fixed $SIG_0$-algebra $A_0$, called *data value algebra*, i.e. $A_0 \in Alg(SIG_0)$.
*Remark:*
For sake of simplicity we consider in this paper only partial or total algebras as in [Bur86] or [EM85], but it is also possible to have an institution independent approach based on institutions in [GB84].

### *3.2 Layer 2: Data States and Transformations*

In layer 2 of the integration paradigm data states and data state transformations of the system are modeled. This may include also states and transformations which are not realizable by the processes of the system from given initial states. The *data state signature* $SIG$ is an extension of the data value signature $SIG_0$ of layer 1, i.e. $SIG_0 \subseteq SIG$, leading to the *layered data state signature*

$$S\text{-}SIG = (SIG_0, SIG)$$

A *data state* is a $SIG$-algebra $A$ such that the restriction $A|_{SIG_0}$ is equal to $A_0$ in layer 1, i.e. $A|_{SIG_0} = A_0$. The class of all data states is denoted by $Mod(S\text{-}SIG)$ and the class of data states $DS$ of the system is a subclass of $Mod(S\text{-}SIG)$, i.e. $DS \subseteq Mod(S\text{-}SIG)$.In order to define data state transformations, layer 2 includes a family $T$ of *transformation symbols* given by

$$T = (T_{v;w})_{(v,w)} \in S^* \times S^*,$$

where $S$ are the sorts of the date state signature $SIG$. For each transformation symbol $t \in T_{v,w}$, written $t : v; w$, we call $v = s_1...s_n \in S^*$ the *input parameter sorts* and $w = s'_1...s'_m \in S^*$ the *output parameter sorts*. The *data state transformation signature* $T\text{-}SIG$ is given by

$$T\text{-}SIG = (S\text{-}SIG, T).$$

A *transformation expression* $t_{A,B}(a, b)$ for $t : v; w$ in $t$ is built up from data states $A, B \in DS$ with *input values* $a \in A_v$ and *output values* $b \in B_w$, where $a \in A_v$ means $a = (a_1, ...a_n) \in A_{s_1} \times ... \times A_{s_n}$ for $v = s_1...s_n$. The set of all transformation expressions for $t : v; w$ is given by

$$t\text{-}EXP = \{t_{A,B}(a, b) | A, B \in DS, a \in A_v, b \in B_w\},$$

and $T\text{-}EXP$ denotes the set of all transformation expressions, i.e.

$$T\text{-}EXP = \bigcup_{t \in T} t\text{-}EXP.$$

The data state transformations of a system $DSTS$ can be defined by a

*transformation condition*

$$cond : T\text{-}EXP \longrightarrow \{true, false\},$$

where $cond(T_{A,B}(a,b)) = true$ means that the transformation $t_{DSTS}$ is *applicable* to data state $A$ with input parameters $a \in A_v$ leading to data state $B$ with output parameters $b \in B_w$. In this case

$$t_{A,B}(a,b) : A \longrightarrow B$$

is called a *transformation step* and the *data state transformation* $t_{DSTS}$ is defined by

$$t_{DSTS} = \{t_{A,B}(a,b) \in t\text{-}EXP | cond(t_{A,B}(a,b)) = true\}$$

for a given transformation condition *cond*.Vice versa $t_{DSTS}$ can be defined as subset of *t-EXP* leading to the transformation condition defined by

$$cond(t_{A,B}(a,b)) = true \Longleftrightarrow t_{A,B}(a,b) \in t_{DSTS}$$

.

A *data state transformation system* $DSTS$ for the data state transformation signature $T\text{-}SIG = (S\text{-}SIG, T)$ is given by

$$DSTS = (A_0, DS, TR),$$

where $DS$ is a class of data states $DS \subseteq Mod(S\text{-}SIG)$ with fixed algebra $A_0$ of layer 1 and $TR$ is a family of data state transformations

$$TR = (t_{DSTS})_{t \in T}.$$

The class of all data state transformation systems $DSTS$ for $T\text{-}SIG$ is denoted by $Mod(T\text{-}SIG)$. The *data state transition* system $G_{DSTS}$ of $DSTS$ is the following graph (resp. transition system)

$$G_{DSTS} = (DS, TS, source, target),$$

where the data states $DS$ are the nodes (resp states) and the edges (resp transitions) are given by

$$TS = \{t_{A,B}(a,b) \in T\text{-}EXP | cond(t_{A,B}(a,b)) = true\}$$

with $source(t_{A,B}(a,b)) = A$ and $target(t_{A,B}(a,b)) = B$.

*Remarks*

(i) Given a specification formalism for transformations based on suitable transformation rules the transformation condition is given by the applicability of these rules and a transformation step above corresponds to a transformation step in this formalism. In this case we obtain the data state transition system $G_{DSTS}$ from $DSTS$ as defined above.

(ii) Vice versa we may assume to have a transition system $G$ with $DS$ as states and a set $TS$ of transitions labeled over *T-EXP*. Then the transformation condition *cond* can be defined by $cond(t_{A,B}(a,b)) = true$ if and only if there is a transition $t$ in $G$ from $A$ to $B$ wiht label $t_{A,B}(a,b) \in$ *T-EXP*. In this case we obtain a data state transformation system DSTS such that we have $G = G_{DSTS}$.

(iii) For several specification formalisms we have for each transformation step $t_{A,B}(a,b) : A \to B$ also a *tracking map track* : $A \to B$, which is a family of partial functions or relations $track_s : A_s \to B_s (s \in S)$.

### *3.3 Layer 3: Processes*

Layer 3 provides the reactive behavior of the system in terms of processes. Processes are the activities of the system according to its aims. Either the behavior is given by one – possibly non-deterministic – process or by a set of processes which realize the different scenarios of the system. Process composition and communication mechanism in the sense of concurrent systems will be considered as part of the architectural layer 4 in 3.4 and section 4. Given a data state transformation signature *T-SIG* with data sorts $S$ as in layer 2 we assume to have in layer 3 a family $P$ of *process symbols* given by

$$P = (P_{v;w})_{(v,w)} \in S^* \times S^*.$$

Similar to transformation symbols in layer 2 each process symbol $p \in P_{v;w}$, written $p : v; w$, has *input parameter sorts* $v = s_1..s_n \in S^*$ and *output parameter sorts* $w = s'_1..s'_m \in S^*$.

The *integrated process signature*, short *process signature*, including the data state transformation signature *T-SIG* and the process symbols $P$ is given by

$$P\text{-}SIG = (T\text{-}SIG, P).$$

In order to define a process $p_{RSTS}$ for $p \in P$ we first consider the reactive state transition system

$$G_{RSTS}(p) = (G(p), I(p), F(p))$$

to be a graph $G(p)$ with sets $I(p)$ and $F(p)$ of initial resp. final states, which are nodes of $G(p)$.The connection between the reactive states (nodes in $G(p)$) with data states in the data state transition system $G_{DSTS}$ of layer 2, and transitions in $G(p)$ with those in $G_{DSTS}$ we consider for $p : \lambda; \lambda$ (without parameters) a graph morphism $h(p) : G_{RSTS}(p) \longrightarrow G_{DSTS}$. In the general case of $p : v; w$ we assume to have *input and output parameter sets* $A \subseteq A_{s1} \times .. \times A_{sn}$ for $v = s_1..s_n$ and $B \subseteq B_{s'_1} \times .. \times B_{s'_m}$ for $w = s'_1..s'_m$ and a family $h(p) = [h(p)(a,b)]_{(a,b)\in A\times B}$ of partial graph morphisms

$$h(p)(a,b) : G_{RSTS}(p) \longrightarrow G_{DSTS}.$$

This means that for given input/output parameters $(a, b)$ only a subgraph of $G(p)$, the domain of $h(p)(a, b)$, represents the reactive state transition system for these parameters. Now a *process* $p_{RSTS}$ for $p \in P$ can be defined by

$$p_{RSTS} = (G_{RSTS}(p), h(p)),$$

where $G_{RSTS}(p)$ is the *reactive state transition system* of $p$ and $h(p)$ a family of partial graph morphisms $h(p)(a, b) : G_{RSTS}(p) \longrightarrow G_{DSTS}$ for $(a, b) \in A \times B$. An *integrated reactive state transformation system* $RSTS$ for an integrated process signature $P\text{-}SIG = (T\text{-}SIG, P)$ and a data state transition system $DSTS$ for $T\text{-}SIG$ in layer 2 is given by

$$RSTS = (DSTS, PR)$$

with a family $PR$ of processes given by

$$PR = (p_{RSTS})p \in P.$$

The class of all reactive state transition systems $RSTS$ for $P\text{-}SIG$ is denoted by $Mod(P\text{-}SIG)$.

*Remarks*

(i) The general case of a family $P$ of process symbols $p : v; w$ includes the special case that $P$ consists of one process symbol $p : \lambda; \lambda$ without parameter sorts. This special case makes especially sense for the specification of CCS-processes see example 3.1, where the behavior of the system is given by one CCS-term defining one CCS-process.This may suggest that CCS processes have no parameters. In fact, they have parameters, but the problem with CCS is that the parameters cannot be presented explicitly in the CCS syntax. One of the essential improvements of LOTOS [Bri89] w.r.t. CCS is the possibility to declare and instantiate parameters.

(ii) On the other hand we have a very large set of processes in the case of Petri nets (see 4.2-4.3). In fact all processes of a Petri net $PN$ given by occurrence nets $OCC_i$ and net morphisms $h_i : OCC_i \to PN$ can also be considered as processes in layer 3, where the reactive state transition systems are given by the case graphs of $OCC_i$. In the case of algebraic high level nets the processes can be considered to have input and output parameters according to the data elements on the input and output places of the corresponding occurrence nets.

(iii) A different interpretation of Petri nets in our model allows to consider the entire net as one process (see alternative 1 in 4.2-4.3).

(iv) In the case of attributed graph transformations, each rule together with a corresponding match corresponds to one process, where not only data of the data value algebra $A_0$, but also nodes and edges of the source

and target graph of a transformation step are taken as parameters to denote the matches from the graphs in the graph rule to those in the transformation step.

(v) We assume in general that for the parameter sets $A$ (resp. $B$) the sets $A_s$ (resp. $B_{s,}$) are the domains of some data state algebra $D$ (resp. $E$) where $D = h(p)(a,b)(i)$ for some $i \in I(p)$ (resp. $E = h(p)(a,b)(f)$ for some $f \in F(p)$) such that $a \in A$ (resp. $b \in B$) can be interpreted as input (resp. output) parameters in the initial (resp. final) reactive states of the process.

### *3.4 Layer 4: System Architecture*

Layer 4 provides the architecture of the system. For this purpose construction, composition and communication mechanism for data types, data transformations and processes should be provided for models of the three basic layers in order to allow a horizontal structuring in each layer. The main idea for layer 4, however, is to have a component concept for integrated specifications and models according to layers 1 to 3. Each component has an explicit import and export interface and there is a composition mechanism to combine components by match of corresponding import and export interfaces.In the framework of this paper we assume to have different construction, composition and communication mechanisms as required above. They will be used in the body of a component to construct new data algebras and states as well as new data state transformations and new processes from corresponding imported items.In the following we only present a basic version of the component concept, where import, export and body of a component are given by integrated signature and models according to layers 1 to 3 and the relationship between signatures is given by inclusion.The general version of the component concept including also constraints, specifications and specification morphisms between the different parts of one component as well as import and export of different components is presented in our paper [EO01].

A *basic component* $COMP$ is given by

$$COMP = (IMP, EXP, BOD)$$

where $IMP, EXP$, and $BOD$ are integrated process signatures according to layer 3 of the integration paradigm satisfying

$$IMP \subseteq BOD \supseteq EXP.$$

$IMP$ is called *import*, $EXP$ is called *export*, and $BOD$ *body* of $COMP$.The *semantics* $SEM(COMP)$ of a basic component $COMP$ is given by a partial function, called *model transformation*,

$$SEM(COMP) : Mod(IMP) \to Mod(EXP)$$

from import to export models in the sense of layer 3 based on construction, composition and communication mechanisms discussed above. In [EO01] we discuss how to construct this model transformation for each basic component based on the notion of constructive specification morphisms between import $IMP$ and body $BOD$ as well as (usual) specification morphisms between export $EXP$ and body $BOD$. For each specification morphism $f : SPEC_1 \to SPEC_2$ we assume to have a restriction construction $RESTR_f : Mod(SPEC_2) \to Mod(SPEC_1)$ corresponding to the forgetful functor in the case of algebraic specifications [EM85].The *composition* of basic components

$$COMP_i = (IMP_i, EXP_i, BOD_i) \qquad (i = 1, 2)$$

is defined if the *match condition*

$$IMP_1 \subseteq EXP_2$$

is satisfied. In this case the composition

$$COMP_3 = COMP_1 \circ COMP_2$$

is a (non-basic) component $COMP_3$ given by

$$COMP_3 = (IMP_3, EXP_3, BOD_3)$$

with

- $IMP_3 = IMP_2$
- $EXP_3 = EXP_1$
- $BOD_3 = (IMP_1, BOD_1, BOD_2)$

where $BOD_1 \supseteq IMP_1 \subseteq BOD_2$.

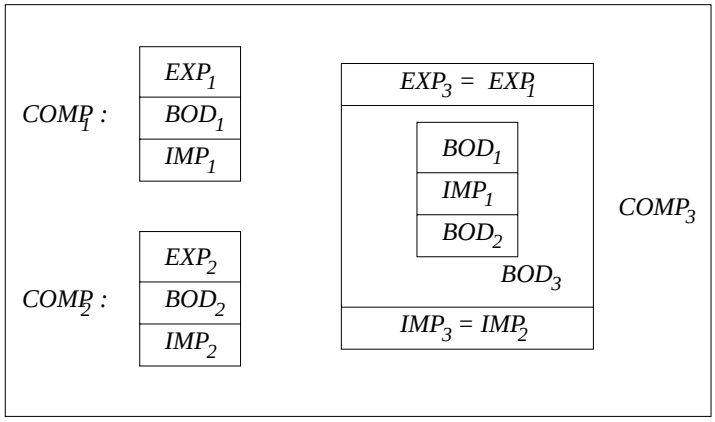


Fig. 1. Composition of Components

The match condition $IMP_1 \subseteq EXP_2$ means that there is a specification morphism from $IMP_1$ to $EXP_2$, which implies that by assumption there is a restriction construction

$$RESTR : Mod(EXP_2) \to Mod(IMP_1)$$

from $EXP_2 - models$ to $IMP_1 - models$. This kind of composition is called *structured composition* in [EO01], because $BOD_3$ is not a basic signature, but a structured one, consisting of $IMP_1, BOD_1$ and $BOD_2$. In [EO01] we discuss how to construct a flat body $BOD_3'$ as gluing (pushout) of $BOD_1$ and $BOD_2$ via $IMP_1$. This leads (up to isomorphism) to a *strong composition* $COMP_3' = (IMP_3, EXP_3, BOD_3')$ which is again a basic component. Structured as well as strong composition can be iterated in a straightforward way.

## 4 Instantiation of the Formal Model

In this section we instantiate the formal model of our integration paradigm by several explicit integrated data type and process formalisms. In principal we could take all examples of our integration paradigm given in our conceptual papers [EO98b,EO00].In fact, these examples include basic techniques, like algebraic specification [EM85], Z [Spi92], UML class diagrams [UML00], Petri nets [Rei85], CCS [Mil89], graph transformation [Roz97], and statecharts [HG96], as well as integrated techniques, like algebraic high level nets [PER95], LOTOS [Bri89], attributed graph transformation [LKW93], and an integration of Z and statecharts [BGK98].In this section we can only consider some of these examples. We show how place/transition and also algebraic high level nets can be considered in two different styles as examples of our formal model. In style 1 transitions of Petri nets are considered as transformation symbols in layer 2, such that the date states are the different markings of the Petri net. In style 2 the data states in layer 2 are black or colored token only, and the Petri net is modeled in layer 3. Moreover we show that there are also different alternatives for layer 3 in each of these cases. Similar to algebraic high level nets also attributed graph transformations can be considered in different styles and alternatives for layer 3, but only style 1 is considered in more detail. While place/transition nets are usually considered as a pure process specification formalism, also pure data type formalisms, like algebraic specifications, in [EM85], can be considered in our framework. In this special case we could have trivial layers 1 and 3, with empty data value specification, and empty sets of transformation and process symbols in layers 2 and 3. Up to now we have discussed how to instantiate our framework by integrated formalisms. But we can also instantiate the framework by explicit examples within one specification formalism. As a typical example we take the CCS sender specification in [Mil89], but also examples based on other techniques like VDM [Jon86], Z [Spi92], and B [Abr96] could be considered.

### *4.1 CCS Sender Specification*

According to [Mil89] we consider the following sender specification, which in this terminology is borrowed from [Gro00] as an example of transformation systems slightly adapted to our framework: The CCS agent $S = S_0, d_0$ is specified by

$$S_{b,d} = \overline{send}(b,d).S'_{b,d}$$

$$S'_{b,d} = \tau.S_{b,d} + ack(b).accept(x).S_{\neg b,x} + ack(\neg b).S'_{b,d},$$

where the index $(b,d)$ of the sender agent $S_{b,d}$ represents the actual state $b$ of the control bit and the message $d$ to be sent. The behavior of the sender agent $S = S_{0,d_0}$in the sense of CCS is given by the transition graph in Figure 2, where $\mathbb{B} = \{t,f\}$ and $D$ is a given set of data or messages with $b \in \mathbb{B}$ and $d, d_0 \in D$.

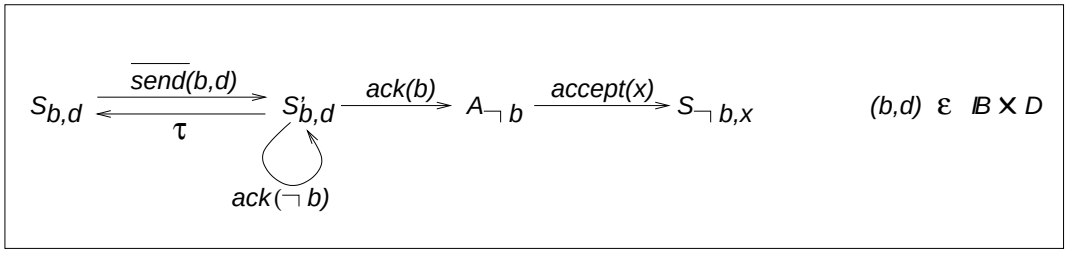


Fig. 2. Behavior of sender agent $S$ in the sense of CCS

In the framework of our integration paradigm this example can be presented in the following way:

*Layer 1 (Data Types of CCS Spec)*
The data value signature $SIG_0$ is given by

$$SIG_0 = sorts : message, bool$$
$$opns : t, f :\rightarrow bool$$
$$\neg : bool \rightarrow bool$$

with $BOOL = (\mathbb{B}, t, f, \neg)$ and the following data value $SIG_0$-algebra

$$A_0 = (D, BOOL)$$

*Layer 2 (Data States and Transformations of CCS Spec)*
The data state signature $SIG$ extending $SIG_0$ is given by

$$SIG = SIG_0 +$$
$$opns : cbit :\rightarrow bool$$
$$msg :\rightarrow message$$

with the following class $DS$ of data states:

$$DS = \{A(b), A(b,d) | b \in I\!B\ , d \in D\}$$

where $A(b) = (D, BOOL, b, undef)$ and $A(b,d) = (D, BOOL, b, d)$ are partial $SIG$-algebras extending $A_0$.The transformation symbols $T$ are given by:

$$\begin{aligned} T = trafos : send : &\quad bool \quad message; \lambda \\ ack : &\quad bool; \lambda \\ accept : &\quad message; \lambda \\ time\text{-}out : &\quad \lambda; \lambda \end{aligned}$$

This leads to the following data state transformation signature

$$T\text{-}SIG = (SIG_0, SIG, T)$$

The data state transformations $TR$ are defined by the data state transition system $G_{DSTS}$ in Figure 3 where $t_{A_1,B_2}(a,b)$ is represented by $t(a,b) : A_1 \to B_2$.

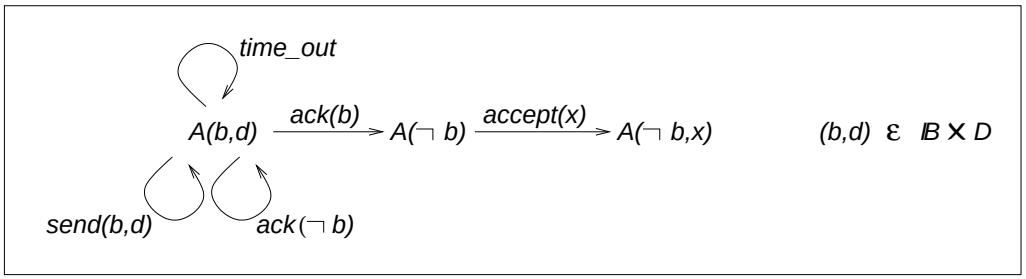


Fig. 3. Data state transition system $G_{DSTS}$

Explicitly $send_{DSTS}$ and $ack_{DSTS}$ are given by:

$$\begin{aligned} send_{DSTS} = &\{send(b,d) : A(b,d) \longrightarrow A(b,d) | (b,d) \in I\!B\ \times D\} \\ ack_{DSTS} = &\{ack(b) : A(b,d) \longrightarrow A(\neg b) | (b,d) \in I\!B\ \times D\} \\ &\cup \{ack(\neg b) : A(b,d) \longrightarrow A(b,d) | (b,d) \in I\!B\ \times D\} \end{aligned}$$

This leads to the following data state transformation system

$$DSTS = (A_0, DS, TR).$$

*Layer 3 (Processes of CCS Spec)*
Since the behavior is specified by one agent we only have one process symbol

$$P = procs : sender : \lambda; \lambda$$

This leads to the integrated process signature

$$P\text{-}SIG = (T\text{-}SIG, P).$$

The corresponding process $sender_{RSTS}$ is given by

$$sender_{RSTS} = (G_{RSTS}(sender), h(sender)),$$

where the reactive state transition system $G_{RSTS}(sender)$ - almost equal to CCS behavior of agent $S$ above - is given by the upper part in Figure 4 with initial state $S_0, d_0$ and empty set of final states, and $h(sender)$ is the graph morphism in Figure 4. Note that the reactive states $S_{b,d}$ and $S'_{b,d}$ have the same data state $A(b, d)$, but different reactive behavior.

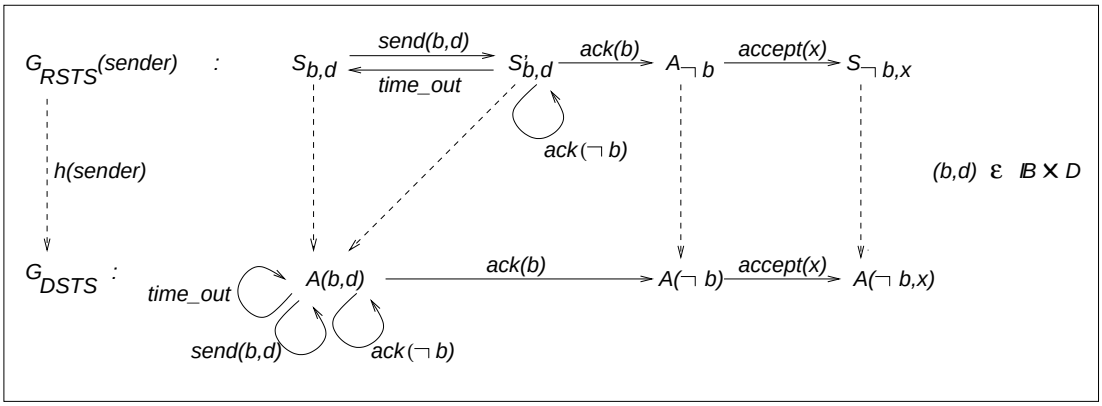


Fig. 4. Process $sender_{RSTS}$

Since we have only one process the processes PR are given by

$$PR = \{sender_{RSTS}\}$$

leading to the following integrated reactive state transformation system

$$RSTS = (DSTS, PR)$$

*Layer 4 (System Architecture of CCS Spec)*
The integrated process signature *P-SIG* above leads to a basic component $COMP = (\phi, P\text{-}SIG, P\text{-}SIG)$, where the semantics is given by the export *P-SIG* model $RSTS$. In general the body of a basic component may include the construction of new processes by parallel composition of imported processes.

### *4.2 Place/Transition Nets*

Let $PN = (Pl, Tr, pre, post, init)$ be a place/transition net with places $Pl$, transitions $Tr$, pre- and post functions

$$pre, post : Tr \to Pl^{\oplus},$$

where $Pl^{\oplus}$ is the free commutative monoid over $Pl$, and $init \in Pl^{\oplus}$ an initial marking (see [MM90] for this algebraic presentation of Petri nets, which is equivalent to the classical presentation in [Rei85]).In order to represent Petri nets in our integration paradigm we could distinguish two different styles. In style 1 data states are markings of the net, while in style 2 data states are black token in the low level and colored token in the high level case. In the following, we only consider style 1.

*Layer 1 (Data Types for P/T Nets)*
The data value signature $SIG_0$ is the signature $nat_0$ of natural numbers

$$\begin{aligned} SIG_0 = nat_0 = \; & sorts : nat \\ & opns : \; zero \to nat \\ & \qquad succ : nat \to nat \end{aligned}$$

and the data value algebra is the algebra of natural numbers $NAT = (I\!N, 0, +1)$.

*Layer 2 (Data States and Transformations for P/T Nets)*
The data state signature $SIG$ is given by

$$\begin{aligned} SIG = \; & SIG_0 + \\ & opns : \quad p :\to nat \quad (p \in Pl) \end{aligned}$$

with the following class of data states

$$DS = \{M = (NAT, (p_M)_{p \in Pl}) | p_M \in I\!N \},$$

where each algebra $M \in DS$ with $(p_M)_{p \in Pl}$ can be considered as a marking $m_M \in Pl^{\oplus}$ defined by

$$m_M = \sum_{p \in Pl} p_M \cdot p$$

and vice versa. The transformation symbols $T$ are given by

$$T = trafos : \quad t : \lambda; \lambda \quad (t \in Tr),$$

where $Tr$ is the set of transitions of $PN$. This leads to the following data state transformation signature

$$T\text{-}SIG = (SIG_0, SIG, T).$$

For each $t \in T$ that data state transformation $t_{DSTS}$ is given by

$$t_{DSTS} = \{t_{M_1,M_2} / M_1, M_2 \in DS \quad and \quad cond(t_{M_1,M_2}) = true\}$$

where $cond(t_{M_1,M_2}) = true$ if

$$pre(t) \leq m_{M_1} \quad and \quad m_{M_2} = m_{M_1} - pre(t) + post(t).$$

In other words, $cond(t_{M_1,M_2}) = true$ means that transition $t$ is activated in marking $m_{M_1}$ and firing of $t$ leads to marking $m_{M_2}$. This leads to the following data state transition system

$$DSTS = (A_0, DS, TR)$$

with $TR = (t_{DSTS})_{t \in T}$ and the data state transition system

$$G_{DSTS} = (DS, TS, source, target)$$

with

$$TS = \{t_{M_1,M_2}/M_1, M_2 \in DS, t \in Tr, cond(t_{M_1,M_2}) = true\}$$

$$source(t_{M_1,M_2}) = M_1 \quad \text{and} \quad target(t_{M_1,M_2}) = M_2.$$

This means that $G_{DSTS}$ is the marking graph of the net $PN$.

*Layer 3 (Processes for P/T Nets)*

**Alternative 1**

In alternative 1 the net $PN$ is considered as one process only given by the process symbol $net$

$$P : procs : net : \lambda, \lambda$$

leading to the integrated process signature $P\text{-}SIG = (T\text{-}SIG, P)$. The corresponding process $net_{RSTS}$ is given by

$$net_{RSTS} = (G_{RSTS}(net), h(net)),$$

where the reactive state transition system $G_{RSTS}(net)$ is the case graph of PN (i.e. the subgraph of the marking graph reachable from the initial marking $init$) and $h(net)$: $G_{RSTS}(net) \rightarrow G_{DSTS}$ is the embedding of the case graph into the marking graph of PN. The integrated reactive state transition system $RSTS$ is given by

$$RSTS = (DSTS, \{net_{RSTS}\})$$

**Alternative 2**

In alternative 2 the processes in layer 3 are exactly the processes of $PN$ in the sense of Petri nets. In this case, net processes are given by occurrence nets $OCC(i)$ together with a net morphisms $m(i) : OCC(i) \rightarrow PN \quad (i \in I)$. The corresponding processes $occ(i)_{RSTS}$ are given by

$$occ(i)_{RSTS} = (G_{RSTS}(occ(i)), h(occ(i))),$$

where the reactive state transition system $G_{RSTS}(occ(i))$ is the case graph of $OCC(i)$ and $h(occ(i)) : G_{RSTS}(occ(i)) \to G_{DSTS}$ is the morphism of the case graph into the marking graph $G_{DSTS}$ of PN induced by $m(i) : OCC(i) \to PN$. The integrated reactive state transition system $RSTS$ in this case is given by $RSTS = (DSTS, PR)$ with

$$PR = \{occ(i)_{RSTS} | i \in I\}$$

The set $I$ above corresponds to the set of all processes of PN. In view of software engineering it makes also sense to consider only a subset $I' \subseteq I$, which represents a set of essential scenarios of the system. Especially $I'$ may be the set of all deterministic processes only.

*Layer 4 (System Architecture for P/T Nets)*
Construction mechanisms for Petri nets to be considered in layer 4 are especially union and fusion. In the following we show how to represent the fusion of nets by a fusion component $COMP_3$, which includes integrated process signatures $P\text{-}SIG_1$ and $P\text{-}SIG_2$ of two arbitrary Petri nets $PN_1$ and $PN_2$ represented by components $COMP_1$ and $COMP_2$. We compose these three components leading to a new component $COMP_4$, which represents the fusion of $PN_1$ and $PN_2$.In more detail we have the following situation, where in the most basic case of fusion we have two Petri nets, $PN_1$ and $PN_2$ together with fusion places $F_1$ of $PN_1$, $F_2$ of $PN_2$, and a bijection $f : F_1 \xrightarrow{\sim} F_2$. The fusion $PN_3$ of $PN_1$ and $PN_2$ via $f$, written $PN_3 = PN_1 +_f PN_2$, is the net $PN_3$ obtained from the disjoint union of $PN_1$ and $PN_2$ by identification of corresponding fusion places. The given nets $PN_1$ and $PN_2$ can be considered for $i = 1, 2$ as basic components

$$COMP_i = (\emptyset, P\text{-}SIG_i, P\text{-}SIG_i),$$

where $P\text{-}SIG_i$ is the integrated process signature for $PN_i$ as defined in layer 3 above with semantics $SEM(COMP_i)$ given by the integrated reactive state transition system $RSTS_i$. The fusion construction of two Petri nets $PN_1$ and $PN_2$ discussed above can be represented by a basic component

$$COMP_3 = (P\text{-}SIG_1 + P\text{-}SIG_2, BOD_3, P\text{-}SIG_3),$$

where $P\text{-}SIG_3$ is the integrated process signature for $PN_3 = PN_1 +_f PN_2$, and

$$
\begin{aligned}
BOD_3 &= (SIG_0, SIG_{123}, T_{123}, P_{123}) \\
SIG_{123} &= SIG_0 + \\
&\qquad opns : p :\to nat \quad (p \in Pl_1 + Pl_2 + Pl_3) \\
T_{123} &= trafos : t : \lambda; \lambda \quad (t \in Tr_1 + Tr_2 + Tr_3) \\
P_{123} &= procs : net_1 : \lambda; \lambda \\
&\qquad\qquad net_2 : \lambda; \lambda \\
&\qquad\qquad net_3 : \lambda; \lambda
\end{aligned}
$$

By constructions we have natural embeddings from the disjoint union $P\text{-}SIG_1 + P\text{-}SIG_2$(with shared $SIG_0$) into $BOD_3$ and from $P\text{-}SIG_3$ into $BOD_3$, where the construction of $P_{123}$ corresponds to alternative 1 in layer 3. The semantics of $COMP_3$ is intended to be a model transformation

$$SEM(COMP_3) : Mod(P\text{-}SIG_1 + P\text{-}SIG_2) \to Mod(P\text{-}SIG_3),$$

which transforms the disjoint union $RSTS_1 + RSTS_2$ of integrated reactive state transition systems $RSTS_1$ and $RSTS_2$ of Petri nets $PN_1$ and $PN_2$ corresponding to $P\text{-}SIG_1$ and $P\text{-}SIG_2$ into the integrated reactive state transition system $RSTS_3$ of the fusion net $PN_3 = PN_1 +_f PN_2$. It remains open to give an explicit partial function on $(P\text{-}SIG_1 + P\text{-}SIG_2)$-models to realize this intended model transformation.

In order to compose the components $COMP_1$ and $COMP_2$ with $COMP_3$ we first construct the disjoint union $COMP_1 + COMP_2$ (with shared $SIG_0$), which has $P\text{-}SIG_1 + P\text{-}SIG_2$ in the export and in the body, and $\emptyset$ in the import. Now the disjoint union $COMP_1 + COMP_2$ can be composed with $COMP_3$ leading to a (non-basic) component

$$COMP_4 = COMP_3 \circ (COMP_1 + COMP_2)$$

with empty import $IMP_4$ and $EXP_4 = EXP_3 = P\text{-}SIG_3$.

The strong composition $COMP_4'$ is given in this case by

$$COMP_4' = (\emptyset, P\text{-}SIG_3, BOD_3)$$

### *4.3 Algebraic High-Level Nets*

Algebraic high-level nets, short AHL nets, have been discussed on a conceptual level already in section 2. In the following we give the main ideas how to model them in our formal framework of section 3, where we now allow specifications instead of signatures in layers 1-4 for the data type and data state parts.

An algebraic high-level net N according to the notation of our paper [PER95] consists of an algebraic specification $SPEC_0$, a $SPEC_0$-algebra $A$, places $Pl$, transitions $Tr$, pre- and post- functions, and an eqns-function assigning to each transition $t \in Tr$ sets $pre(t), post(t)$, and $eqns(t)$ respectively. As shown in figure 6, the arc-inscriptions of a high-level net $N$ for a transition

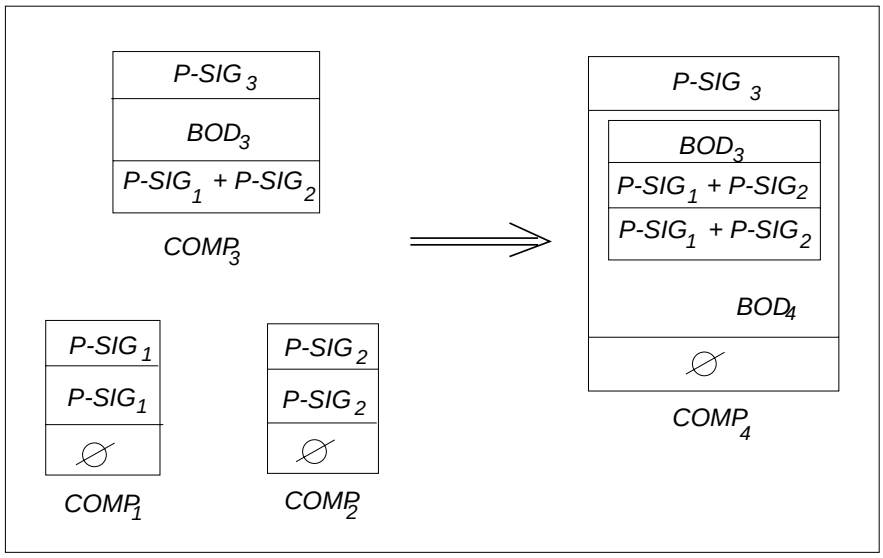


Fig. 5. Composition of Petri Net Components

$t \in Tr$ wiht places $p_1, ...p_n \in Pl$ in the pre-domain and $p_1', ...p_m' \in Pl$ in the post-domain of $t$ are given by multiset terms $term_i$ and $term_j$ with variables over the signature $SIG_0$ of $SPEC_0$. The node-inscription of $t$ is a set $E$ of equations over $SIG_0$.

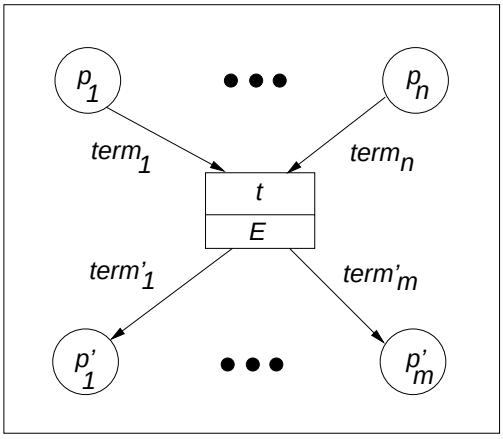


Fig. 6. Transition with pre- and post-domain

In figure 6 the sets $pre(t)$, $post(t)$ and $eqns(t)$ are given by

$$pre(t) = (term_1, p_1) \oplus .. \oplus (term_n, p_n),$$
$$post(t) = (term_1', p_1') \oplus .. \oplus (term_m', p_m'), \quad \text{and}$$
$$eqns(t) = E.$$

In our framework it is useful to extend algebraic high-level nets $N$ in the sense of [PER95] by a type-function assigning to each place $p \in Pl$ a sort $type(p) \in S$, called type of $p$, and an initial marking $init$ consisting of a multiset of data token on some places. A data token $d$ on $p \in Pl$ is an element $d \in A_s$ of the algebra $A$ with $s = type(p)$.

*Layer 1 (Data Types for AHL Nets)*
The data value specification $SPEC_0$ is the algebraic specification $SPEC_0$ of $N$ and the data value algebra is the $SPEC_0$ -algebra $A$ of $N$.

*Layer 2 (Data States and Transformations for AHL Nets)*
The data state specification $SPEC$ is given by a multiset extension of $SPEC_0$ with additional constant symbols $p :\rightarrow mult(type(p))$ for each $p \in Pl$, where $mult(type(p))$ is the multisort of sort $type(p)$. The class $DS$ of data states is given by all those $SPEC$-algebras $M$, where the $SPEC_0$-part is equal to $A$. This means that there is a bijective correspondence between data states $M$ and markings $m$ of the high level net $N$. For each transition $t \in Tr$ we have exactly one transformation symbol $t : v; \lambda$, where $v = s_1..s_n$ are the sorts $s_i$ of the variables of the terms occurring in $pre(t), post(t)$ and $eqns(t)$.

This leads to the following data state transformation specification

$$T\text{-}SPEC = (SPEC_0, SPEC, T),$$

where $T$ is the set of all transformation symbols. Moreover, we obtain data state transformation and transition systems

$$DSTS = (A, DS, T_{DSTS}) \qquad \text{and} \qquad G_{DSTS} = (DS, TS, source, target)$$

where $G_{DSTS}$ coincides – up to isomorphism – with the marking graph of the net $N$.

*Layer 3 (Processes for AHL Nets)*
Similar to the low-level case of place/transition nets we can consider two alternatives. In alternative 1 the net 1 is embedded into the marking graph $G_{DSTS}$ of $N$. In alternative 2 the processes are defined as high-level processes of $N$, which are high-level occurrence nets $OCC(i)$ with morphisms $m(i) : OCC(i) \rightarrow N$. A theory of such high-level processes of algebraic high-level nets is under development by the authors. The corresponding processes of the reactive state transition system $RSTS$ would be morphisms from different initial markings into the marking graph $G_{DSTS}$ of $N$.

*Layer 4 (System Architecture for AHL Nets)*
The general ideas presented in layer 4 of place/transition nets above can in principle be extended to algebraic high level nets, but it remains open to discuss corresponding high-level constructions and a suitable component concept in more detail.

### *4.4 Attributed Graph Transformation*

Attributed graphs and attributed graph transformation systems, short AGTs, have been discussed on a conceptual level already in section 2. In the following we give the main ideas how to model them in our formal framework of section 3, where – similar to 4.3 – we allow specifications instead of signatures in layers 1 - 4 for the data type and data state parts.

An attributed graph structure specification $ATTR$ consists of a graph structure signature, which is an algebraic signature with unary operation symbols only, an algebraic data type specification $SPEC_0$ with sorts $S_0$, and attribute assignment functions from each graph signature sort $s$ to a suitable data type sort $s_0 \in S_0$.

An attributed graph transformation system $AGT$ according to [ELO95] and [LKW93] consists of an attributed graph structure specification $ATTR$, a set of start graphs, and a set of transformation rules $t : L \to R$, where $L$ and $R$ are $ATTR-algebras$. Given a rule $t : L \to R$ and a match $m_L : L \to G$ an attributed graph transformation $G \Rightarrow H$ is defined, where $H$ is obtain form $G$ by replacing $L$ in $G$ by $R$. Formally this can be expressed by pushouts in suitable categories of attributed graph structures given in [LKW93,ELO95].

*Layer 1 (Data Types for AGTs)*
The data value specification $SPEC_0$ is the algebraic data type specification of $ATTR$ and the data value algebra is a given $SPEC_0$-algebra $A_0$.

*Layer 2 (Data States and Transformations for AGTs)*
The data state specification $SPEC$ is given by the attributed graph structure specification $ATTR$ of $AGT$. The class $DS$ of data states is given by all $ATTR$-algebras $A$ which are extensions of the data value algebra $A_0$. For each rule $t : L \to R$ of $AGT$ we have a transformation symbol $t : s_1...s_n; \lambda$, where $s_1...s_n$ are the sorts of the data type variables occurring in the attributes of $L$ and $R$. As shown in [ELO95], we can assume that an assignment of these variables in a graph $G$ is sufficient to determine a match $m : L \to G$ of the rule in $G$. Let $T$ be the set of all these transformation symbols $t : s_1..s_n; \lambda$, then the data state transformation specification $T\text{-}SPEC$ is given by

$$T\text{-}SPEC = (SPEC_0, ATTR, T).$$

The transformations of the $AGT$ are defining a data state transformation system $DSTS$ and a data state transition system $G_{DSTS}$, which is the graph of all transformations of the attributed graph transformation system $AGT$.

*Layer 3 (Processes for AGTs)*
Similar to Petri nets we can consider two alternatives. In alternative 1 the attributed graph transformation system $AGT$ is considered as one process only. In alternative 2 the processes on layer 3 are defined by suitable graph

processes for *AGT* similar to those in [CMR96]. As proposed in [EB94] we can consider processes with parameters in layer 3 which are built up as transactions of actions defined by specific transformation sequences in layer 2.

*Layer 4 (System Architecture for AGTs)*
There are several construction mechanisms for graph transformation rules, like parallel, concurrent, and amalgamated rules, which can be extended as construction mechanisms for attributed graph transformation systems in the sense of layer 4. A generic component concept for graph transformation systems is discussed in [EO01], which is similar to that of generic graph transformation systems in [EE96].

## 5 Conclusion

In this paper we have presented a formal model for our integration paradigm introduced on a conceptual level in our papers [EO98b,EO00]. The formal model for the three basic layers of the integration paradigm is closely related to the notion of transformation systems in [Gro98,Gro00]. The main differences are the following: In our model each transformation step is given by a single transformation expressions corresponding to parallel or synchronized transformation steps. Moreover constructions for transformation systems are given explicitly in [Gro98,Gro00], which can be used as constructions in our layer 4. On the other hand our model allows process signatures and hence a family of processes in layer 3 in contrast to one main process in a transformation system of [Gro98,Gro00]. In our model we have considered in this paper only signatures and models corresponding to signatures. This model is extended in [EO01] by constraints for signatures in the sense of sentences in institutions [GB84] for each layer. This allows considering model specifications not only in the three basic layers but also in import, export and body of our component concept. This is shown in our instantiations by algebraic high level nets and attributed graph transformations in this paper already. The concept of components with model specifications instead of signatures becomes much more powerful, because constraints in the import can be considered as requirements for the import while those in the export as properties of the component. This has been shown already for algebraic module specifications in [EM90]. In [EO01] we discuss how essential parts of the theory of algebraic module specifications in [EM90] can be extended to components for integrated data type and process specifications. Several interesting concepts and examples for the specification of properties and requirements and for the construction of transformation systems are given already in [Gro98,Gro00,Gro01] which are promising to be used in our model. Instantiations of our formal model have been given for different kinds of Petri nets and graph transformations and a specific CCS example in this paper. It remains open to study also instantiations for all the other examples of our conceptual model discussed in

[EO98b,EO00], especially for different diagram techniques in UML [UML00].

## References


[Abr96] J.R. Abrial. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996.

[AZ95] E. Astesiano and E. Zucca. D-oids: A model for dynamic data types. *Math. Struct. in Comp. Sci.*, 5(2):257–282, 1995.

[BGK98] R. Büssow, R. Geisler, and M. Klar. Specifying safety-critical Embedded Systems with Statecharts and Z: A Case Study. *Springer LNCS 1382*, pages 71 – 87, 1998.

[Bri89] Brinksma, E. (ed.). Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour. ISO 8807, 1989. International Standard.

[Bur86] P. Burmeister. *A Model Theoretic Oriented Approach to Partial Algebras*, volume 32 of *Mathematical Research — Mathematische Forschung*. Akademie-Verlag, Berlin, 1986.

[CMR96] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Special Issue of Fundamenta Informaticae*, 26(3,4):241–266, 1996.

[Cor98] F. Cornelius. *A Semantical Reference Model for the Integration of Different Dimensions of Distributed System Specifications*. PhD thesis, Technische Universität Berlin, 1998.

[DG94] P. Dauchy and M.C. Gaudel. Algebraic specifications with implicit states. *Tech. Report, Univ. Paris Sud*, 1994.

[DRS94] R. Duke, G. Rose, and G. Smith. Object-Z: a Specification Language for the Description of Standards. Technical Report 94–45, Software Verification Research Centre, Department of Computer Science, The University of Queensland, Australia, 1994.

[EB94] H. Ehrig and R. Bardohl. Specification Techniques using Dynamic Abstract Data Types and Application to Shipping Software. In *Proc. of the International Workshop on Advanced Software Technology*, pages 70–85, 1994.

[EBC+96] H. Ehrig, R. Bardohl, F. Cornelius, R. Geisler, Große-Rhode, and J. Padberg. A new integration paradigm for formal specification of safe software systems. In *Proc. 10th Japan-Germany Forum on Information Technology*. Gesellschaft für Mathematik und Datenverarbeitung, 1996.

[EE96] H. Ehrig and G. Engels. Pragmatic and semantic aspects of a module concept for graph transformation systems. In *LNCS 1073 , Proc. Williamsburg, U.S.A.*, pages 137–154. Springer Verlag, 1996.

[ELO95] H. Ehrig, M. Löwe, and F. Orejas. Dynamic abstract data types based on algebraic graph transformations. In *Proc. ADT-Workshop '94, Springer LNCS, 906*, pages 236–254, 1995.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1985.

[EM90] 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.

[EO94] H. Ehrig and F. Orejas. Dynamic abstract data types: An informal proposal. *Bull. EATCS 53*, pages 162–169, 1994. also in [PRS01], pages 180 - 191.

[EO98a] H. Ehrig and F. Orejas. Integration and classification of data type and process specification techniques. Technical Report 98-10, Technical University of Berlin, 1998.

[EO98b] H. Ehrig and F. Orejas. Integration Paradigm for Data Type and Process Specification Techniques. *Bull. EATCS 65, Formal Specification Column, Part 5*, 1998. also in [PRS01], pages 192 - 201.

[EO00] H. Ehrig and F. Orejas. Integration and Classification of Data Type and Process Specification Techniques, 2000. submitted.

[EO01] H. Ehrig and F. Orejas. A Component Concept for Integrated Data Type and Process Specification Techniques. Technical report, Technische Universität Berlin, FB Informatik, 2001. to appear.

[GB84] J.A. Goguen and R.M. Burstall. Introducing institutions. *Proc. Logics of Programming Workshop, Carnegie-Mellon*, Springer LNCS 164:221 – 256, 1984.

[GKP98] R. Geisler, M. Klar, and C. Pons. Dimensions and Dichotomy in Metamodeling. Technical Report 98–05, FB Informatik, TU Berlin, 1998.

[Gro98] M. Große–Rhode. Algebra transformation systems and their composition. In E. Astesiano, editor, *Fundamental Approaches to Software Engineering (FASE'98)*, pages 107–122. Springer LNCS 1382, 1998.

[Gro99] M. Große–Rhode. On a reference model for the formalization and integration of software specification languages. In *Bulletin of the EATCS No 68*, pages 81–89. 1999.

[Gro00] M. Große–Rhode. Compositional Comparison of Formal Software Specifications using Transformation Systems, August 2000. submitted.

[Gro01] M. Große–Rhode. Semantic Integration of Heterogeneous Formal Specifications via Transformation Systems, 2001. to appear.

[Har87] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HEET99] Reiko Heckel, Hartmut Ehrig, Gregor Engels, and Gabriele Taentzer. Classification and Comparison of Modularity Concepts for Graph Transformation Systems. In H. Ehrig, G. Engels, J.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*, pages 669 – 690. World Scientific, 1999.

[HG96] D. Harel and E. Gery. Executable object modeling with Statecharts. In *IEEE Computer, vol. 30, no. 7*. IEEE, 1996.

[Jon86] C. B. Jones. *Systematic software development using VDM*. Prentice-Hall International, London, 1986.

[Kla99] M. Klar. *A Semantical Framework for the Integration of Object-Oriented Modeling Languages*. PhD thesis, TU Berlin, FB13, 1999.

[LKW93] M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 14, pages 185–199. John Wiley & Sons Ltd, 1993.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[MK98] S. Mann and M. Klar. A metamodel for object-oriented statecharts. In *Proc. 2nd Workshop on Rigorous Object-Oriented Methods, ROOM 2*. University of Bradford, 1998.

[MM90] J. Meseguer and U. Montanari. Petri Nets are Monoids. *Information and Computation*, 88(2):105–155, 1990.

[PER95] J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science*, 5:217–256, 1995.

[PP91] F. Parisi-Presicce and A. Pierantonio. An Algebraic Approach to Inheritance and Subtyping. In *Proc. ESEC 1991, LNCS 550*, pages 364 –379. Springer, 1991.

[PRS01] G. Paun, G. Rozenberg, and A. Salomaa, editors. *Current Trends in Theoretical Computer Science: Entering the 21st Century*. World Scientific, Singapore etc., 2001.

[Rei85] W. Reisig. *Petri Nets*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985.

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.

[UML00] *Unified Modeling Language – version 1.3*, 2000. Available at http://www.omg.org/uml.