*Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) { Proc. GT-VMT 2001 URL:* [*http://www.elsevier.nl/locate/entcs/volume50.html*](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 tech- niques, 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 for- mal 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. Alge- braic 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 speci cation, 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 di erent kinds of data type and process modeling tech-* niques 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 di erent formalisms or jointly by an integrated formal- ism. We roughly distinguish the following classes of data type speci cation and modeling formalisms:

*algebraic/axiomatic approaches,*

*c 2001 Published by Elsevier Science B. V.* Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

*state/model-oriented approaches,*

*class-oriented approaches.*

*The main classes of process speci cation 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 xed data domains of alphabets, and also high level variants that are de ned by integration of basic process and data type modeling techniques. Based on these examples of data type and pro- cess modeling techniques we have introduced in [EO98b,EO00] an integration paradigm for system speci cation and modeling on four di erent layers, which provides a uni ed 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 tech- niques. Our formal model has been in uenced by the semantical reference models for formalization and integration of Cornelius [Cor98] and Gro e-

*Rhode [Gro98,Gro99,Gro00,Gro01]. Especially it is closely related to the no-*

*tion of transformation systems in [Gro98,Gro00] concerning the formal model* of layers 1-3 of our integration paradigm. The formal model for layer 4 (sys- tem 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 sys- tem 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 instanti- ations 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 inte-* grated modeling techniques, especially to Martin Gro e-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 prior- ity program "Integration of Software Speci cation Techniques for Applications in Engineering" and by the Spanish project HEMOSS (TIC 98-0949-C02-01).

# *2 Integration Paradigm for Data Type and Process Spec-* i cation and Modeling Techniques

*As typical examples for the integration of data type and process speci ca-* tion formalisms we consider algebraic high-level nets [PER95], an integra- tion of algebraic speci cation and Petri nets, attributed graph transformation [ELO95], an integration of algebraic speci cation and graph transformation,

*SZ [BGK98], an integration of Z and statecharts, and UML [UML00] com-*

*prising di erent object-oriented modelling techniques.In all these and several* other examples like LOTOS [Bri89], an integration of algebraic speci cation 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 rst layer corresponds exactly to the data type view of the system. The* following layers are integrated views of data type, data state and system archi- tecture aspects, where each layer is based on the previous one. This hierarchi- cal concept is di erent from a dimensions concept (see e.g. [Cor98,GKP98]) where dimensions are considered to be independent and can be combined in di erent 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 commu- nication 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 de ned 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 compo- nents 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 de ned by algebraic speci cations such that the tokens of high-level nets are no longer black tokens, but data tokens as ele- ments of algebras. In layer 2 the data states are given by the markings of places by data tokens and the transformation of states by ring of transitions. In layer 3 we have net processes, which can be de ned 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 speci cations [EM85,EM90], net transformations de ned by rule-based trans- formation 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*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Speci cation 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 de ned by algebraic*  *speci cation* | *marking of places by data tokens & ring*  *of transitions* | *net processes* | *parameterization, net trans- formations*  *union / fusion* |
| *Attributed Graph Trans-*  *formation* | *attributes and algebras de ned by algebraic*  *speci cation* | *attributed graphs and graph*  *transformation* | *graph processes* | *parameterization, composition,*  *modularization* |
| *SZ* | *type*  *de nition in Z* | *data states and*  *operation schemas in Z* | *statechart processes* | *con gurations* |
| *UML* | *basic data types de ned by class diagrams* | *classes, attributes & methods* | *object- oriented statecharts & sequence*  *diagrams* | *packages* |

*2.3 Attributed Graph Transformation*

*Similar to algebraic high-level nets the speci cation formalism of attributed* graph transformations [ELO95] is based on algebraic speci cations 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 de ned 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 spec- i cations and several notions of composition and modularization for graph transformation systems are under development [EE96,HEET99].

## *2.4 SZ*

*The speci cation formalism SZ [BGK98] was developed in the ESPRESS-* project for the speci cation of safety critical embedded systems. The main idea of SZ is to integrate statecharts [Har87] with Z [Spi92] and to have a notion of con gurations as modular structuring mechanism. This means that in the data type layer 1 we have type de nition 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 de ned by statecharts and the system architecture in layer 4 is given by con gurations of SZ.

## *2.5 UML*

*The Uni ed Modelling Language UML [UML00] combines several semi-formal* diagram techniques for object-oriented modelling, where a semantical founda- tion and integration of all these techniques is still missing but under devel- opment in di erent projects. We here only consider some of these diagram techniques. The data types in layer 1 are given by basic data types de ned by class diagrams. Data states and transformations in layer 2 are given by classes de ning the internal states by attributes and state transformations by meth- ods. Processes in layer 3 within one object are de ned using object-oriented statecharts [HG96] while sequence diagrams model processes between di er- ent objects. The package concept is a rst 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 re-* viewed 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 speci cation 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 SIG0 is an algebraic signature SIG0 = (S0; 0) consisting of a set S0 of sorts and a family 0 of operation symbols (see e.g. [EM85]) or a rst order signatureSIG0 = (S0; 0; 0), where

*0 is a family of predicate symbols.On the semantical level we allow to have*

*partial algebras. However, total algebras are suÆcient in several applications.* In layer 1 we assume to have a xed SIG0-algebra A0, called data value algebra,

*i.e. A0 2 Alg(SIG0).*

*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 transforma-* tions of the system are modeled. This may include also states and transfor- mations 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 SIG0 of layer 1, i.e. SIG0 SIG, leading to the layered data state signature

## *S-SIG = (SIG0; SIG)*

*A data state is a SIG-algebra A such that the restriction AjSIG0 is equal* to A0 in layer 1, i.e. AjSIG0 = A0. The class of all data states is denoted by Mod(S-SIG) and the class of data states DS of the system is a subclass of Mod(S-SIG), i.e. DS Mod(S-SIG).In order to de ne data state transfor- mations, layer 2 includes a family T of transformation symbols given by

## *T = (T ) 2 S S ;*

*v;w (v;w)*

*where S are the sorts of the date state signature SIG. For each transfor-* mation symbol t 2 Tv;w, written t : v; w, we call v = s1:::sn 2 S the input

*parameter sorts and w = s; :::s;*

*2 S the output parameter sorts. The data*

*1 m*

*state transformation signature T-SIG is given by*

## *T-SIG = (S-SIG;T ):*

*A transformation expression tA;B(a; b) for t : v; w in t is built up from data* states A; B 2 DS with input values a 2 Av and output values b 2 Bw, where a 2 Av means a = (a1; :::an) 2 As ::: As for v = s1:::sn. The set of all transformation expressions for t : v; w is given by

*1 n*

*t-EXP = ftA;B(a; b)jA; B 2 DS; a 2 Av;b 2 Bwg;*

*and T-EXP denotes the set of all transformation expressions, i.e.*

*T-EXP = [t-EXP:*

*t2T*

*The data state transformations of a system DST S can be de ned by a*

*transformation condition*

*cond : T-EXP ! ftrue; f alseg;*

*where cond(TA;B(a; b)) = true means that the transformation tDSTS is appli-* cable to data state A with input parameters a 2 Av leading to data state B with output parameters b 2 Bw. In this case

*tA;B(a; b) : A ! B*

*is called a transformation step and the data state transformation tDSTS is* de ned by

*tDSTS = ftA;B(a; b) 2 t-EXPjcond(tA;B(a; b)) = trueg*

*for a given transformation condition cond.Vice versa tDSTS can be de ned* as subset of t-EXP leading to the transformation condition de ned by

*cond(tA;B(a; b)) = true () tA;B(a; b) 2 tDSTS*

*.*

*A data state transformation system DST S for the data state transforma-*

*tion signature T-SIG = (S-SIG;T ) is given by*

## *DST S = (A0; DS; T R);*

*where DS is a class of data states DS Mod(S-SIG) with xed algebra* A0 of layer 1 and TR is a family of data state transformations

*TR = (tDSTS)t2T :*

*The class of all data state transformation systems DST S for T-SIG is* denoted by Mod(T-SIG). The data state transition system GDSTS of DST S is the following graph (resp. transition system)

*GDSTS = (DS; T S; source; target);*

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

*TS = ftA;B(a; b) 2 T-EXPjcond(tA;B(a; b)) = trueg* with source(tA;B(a; b)) = A and target(tA;B(a; b)) = B.

*Remarks*

1. *Given a speci cation formalism for transformations based on suitable transformation rules the transformation condition is given by the appli- cability 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 GDSTS from DST S as de ned above.*
2. *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 trans- formation condition cond can be de ned by cond(tA;B(a; b)) = true if and only if there is a transition t in G from A to B wiht label tA;B(a; b)* *2*

*T-EXP. In this case we obtain a data state transformation system DSTS*

*such that we have G = GDSTS.*

1. *For several speci cation formalisms we have for each transformation step tA;B(a; b) : A ! B also a tracking map track : A ! B, which is a family of partial functions or relations tracks : As ! Bs(s 2 S).*
   1. *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 di erent scenarios of the system. Process compo- sition 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 ) 2 S S :*

*v;w (v;w)*

*Similar to transformation symbols in layer 2 each process symbol p* *2*

*Pv;w; written p : v; w, has input parameter sorts v = s1::sn 2 S and output*

*parameter sorts w = s; ::s;*

*2 S :*

*1 m*

*The integrated process signature, short process signature, including the data*

*state transformation signature T-SIG and the process symbols P is given by*

## *P-SIG = (T-SIG;P ):*

*In order to de ne a process pRSTS for p 2 P we rst consider the reactive* state transition system

*GRSTS(p) = (G(p); I(p);F (p))*

*to be a graph G(p) with sets I(p) and F (p) of initial resp. nal 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 GDSTS of layer 2, and transitions in G(p) with those in GDSTS we consider for p : ; (without parameters) a graph morphism h(p) : GRSTS(p) ! GDSTS. In the general case of p : v; w we assume to have input and output parameter sets A

*As1 :: Asn for v = s1::sn*

*and B B ; :: B ; for w = s; ::s;*

*and a*

*family h(p) = [h(p)(a; b)](a;b)2A B of partial graph morphisms*

*s1 sm 1 m*

*h(p)(a; b) : GRSTS(p) ! GDSTS:*

*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 sys- tem for these parameters. Now a process pRSTS for p 2 P can be de ned by

*pRSTS = (GRSTS(p); h(p));*

*where GRSTS(p) is the reactive state transition system of p and h(p) a family* of partial graph morphisms h(p)(a; b) : GRSTS(p) ! GDSTS for (a; b) 2 A B. An integrated reactive state transformation system RST S for an integrated process signature P-SIG = (T-SIG;P ) and a data state transition system DST S for T-SIG in layer 2 is given by

## *RST S = (DST S; P R)*

*with a family PR of processes given by*

*PR = (pRSTS)p 2 P:*

*The class of all reactive state transition systems RST S for P-SIG is de-* noted by Mod(P-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 : ; without param- eter sorts. This special case makes especially sense for the speci cation of CCS-processes see example 3.1, where the behavior of the system is given by one CCS-term de ning 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 OCCi and net morphisms hi : OCCi ! PN can also be considered as processes in layer 3, where the reactive state transition systems are given by the case graphs of OCCi. 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 di erent 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 A0, 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* As (resp. Bs; ) are the domains of some data state algebra D (resp. E) where D = h(p)(a; b)(i) for some i 2 I(p) (resp. E = h(p)(a; b)(f ) for some f 2 F (p)) such that a 2 A (resp. b 2 B) can be interpreted as input (resp. output) parameters in the initial (resp. nal) 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 transforma- tions 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 speci cations and models according to layers 1 to 3. Each component has an explicit im- port 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 di erent construction, composi- tion 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 im- ported 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 inte- grated 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, speci cations and speci cation morphisms between the di erent parts of one component as well as import and export of di erent 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 BOD 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 ) ! Mod(EXP )*

*from import to export models in the sense of layer 3 based on construc-* tion, composition and communication mechanisms discussed above. In [EO01] we discuss how to construct this model transformation for each basic compo- nent based on the notion of constructive speci cation morphisms between import IMP and body BOD as well as (usual) speci cation morphisms be- tween export EXP and body BOD. For each speci cation morphism f : SP EC1 ! SP EC2 we assume to have a restriction construction REST Rf : Mod(SP EC2) ! Mod(SP EC1) corresponding to the forgetful functor in the case of algebraic speci cations [EM85].The composition of basic components

*COMPi = (IMPi; EXPi; BODi) (i = 1; 2)*

*is de ned if the match condition*

## *IMP1 EXP2*

*is satis ed. In this case the composition*

## *COMP3 = COMP1 Æ COMP2*

*is a (non-basic) component COMP3 given by*

## *COMP3 = (IMP3; EXP3; BOD3)*

*with*

*IMP3 = IMP2*

*EXP3 = EXP1*

*BOD3 = (IMP1; BOD1; BOD2)*

*where BOD1 IMP1 BOD2:*

|  |
| --- |
| *EXP1* |
| *BOD1* |
| *IMP1* |

|  |
| --- |
| *BOD1* |
| *IMP1* |
| *BOD2* |

*Fig. 1. Composition of Components*

*COMP1 :*

*COMP3*

*COMP2 :*

|  |
| --- |
| *EXP3 = EXP1* |
| *BOD3* |
| *IMP3 = IMP2* |

|  |
| --- |
| *EXP2* |
| *BOD2* |
| *IMP2* |

*The match condition IMP1 EXP2 means that there is a speci cation* morphism from IMP1 to EXP2, which implies that by assumption there is a restriction construction

*REST R : Mod(EXP2) ! Mod(IMP1)*

*from EXP2 models to IMP1 models. This kind of composition is called* structured composition in [EO01], because BOD3 is not a basic signature, but a structured one, consisting of IMP1; BOD1 and BOD2. In [EO01] we discuss

*how to construct a at body BOD0 as gluing (pushout) of BOD1 and BOD2*

*3*

*via IMP1. This leads (up to isomorphism) to a strong composition COMP 0 =* (IMP3; EXP3; BOD0 ) which is again a basic component. Structured as well as strong composition can be iterated in a straightforward way.

*3*

*3*

# *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 concep- tual papers [EO98b,EO00].In fact, these examples include basic techniques, like algebraic speci cation [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 integra- tion 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 di erent 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 di erent 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 di erent alternatives for layer 3 in each of these cases. Similar to alge- braic high level nets also attributed graph transformations can be considered in di erent 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 speci cation formalism, also pure data type formalisms, like algebraic speci cations, in [EM85], can be considered in our framework. In this special case we could have trivial layers 1 and 3, with empty data value speci ca- tion, 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 speci cation formalism. As a typical example we take the CCS sender speci cation in [Mil89], but also examples based on other techniques like VDM [Jon86], Z [Spi92], and B [Abr96] could be considered.

*4.1 CCS Sender Speci cation*

*According to [Mil89] we consider the following sender speci cation, which in* this terminology is borrowed from [Gro00] as an example of transformation systems slightly adapted to our framework: The CCS agent S = S0; d0 is speci ed by

*Sb;d*

*= send(b; d):S0*

*;*

*S*

*b;d*

## *= :S*

*b;d*

*+ ack(b):accept(x):S*

*:b;x*

*b;d*

*+ ack(:b):S0 ;*

*where the index (b; d) of the sender agent Sb;d represents the actual state b of* the control bit and the message d to be sent. The behavior of the sender agent S = S0;d0 in the sense of CCS is given by the transition graph in Figure 2, where IB = ft; f g and D is a given set of data or messages with b 2 IB and d; d0 2 D.

*b;d*

*Sb,d*

*send(b,d)*



*S’b,d*

*ack(b)*

*A*

*accept(x)*

*b*

*S*

*b,x*

*(b,d)*  *IB* ✕ *D*

*ack*  *b)*

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

*In the framework of our integration paradigm this example can be pre-* sented in the following way:

*Layer 1 (Data Types of CCS Spec)*

*The data value signature SIG0 is given by* SIG0 = sorts : message; bool

*opns : t; f :! bool*

*: : bool ! bool*

*with BOOL = (IB ; t; f; :) and the following data value SIG0-algebra*

## *A0 = (D; BOOL)*

*Layer 2 (Data States and Transformations of CCS Spec)* The data state signature SIG extending SIG0 is given by

## *SIG = SIG0+*

*opns : cbit :! bool*

*msg :! message*

*with the following class DS of data states:*

*DS = fA(b); A(b; d)jb 2 IB ; d 2 Dg*

*where A(b) = (D; BOOL; b; undef ) and A(b; d) = (D; BOOL; b; d) are partial* SIG-algebras extending A0.The transformation symbols T are given by:

*T = traf os : send : bool message;*

*ack : bool; accept : message; time-out : ;*

*This leads to the following data state transformation signature*

## *T-SIG = (SIG0; SIG;T )*

*The data state transformations TR are de ned by the data state transition* system GDSTS in Figure 3 where tA1 ;B2 (a; b) is represented by t(a; b) : A1 ! B2.

*time\_out*

*A(b,d)*

*ack(b)*

*A(*

*b) accept(x)*

*A( b,x)*

*(b,d)*  *IB* ✕ *D*

*send(b,d)*

*ack*  *b)*

*Fig. 3. Data state transition system GDST S Explicitly sendDSTS and ackDSTS are given by:*

*sendDSTS = fsend(b; d) : A(b; d) ! A(b; d)j(b; d) 2 IB Dg*

*ackDSTS = fack(b) : A(b; d) ! A(:b)j(b; d) 2 IB Dg*

*[fack(:b) : A(b; d) ! A(b; d)j(b; d) 2 IB Dg*

*This leads to the following data state transformation system*

## *DST S = (A0; DS; T R):*

*Layer 3 (Processes of CCS Spec)*

*Since the behavior is speci ed by one agent we only have one process symbol*

*P = procs : sender : ;*

*This leads to the integrated process signature*

## *P-SIG = (T-SIG;P ):*

*The corresponding process senderRSTS is given by*

*senderRSTS = (GRSTS(sender); h(sender));*

*where the reactive state transition system GRSTS(sender) - almost equal to* CCS behavior of agent S above - is given by the upper part in Figure 4 with initial state S0; d0 and empty set of nal states, and h(sender) is the graph

*morphism in Figure 4. Note that the reactive states Sb;d and S0 have the*

*b;d*

*same data state A(b; d), but di erent reactive behavior.*

*G (sender)*

*RSTS*

*:*

*Sb,d*

*send(b,d)*

*time\_out*

*S’b,d*

*ack(b)*

*A*

*accept(x)*

*b*

*S*

*b,x*

*ack*  *b)*

*h(sender)*

*(b,d)*  *IB* ✕ *D*

*G*

*DSTS*

*:*

*ack(b)*

*time\_out*

*A(b,d)*

*A(*

*b) accept(x)*

*A( b,x)*

*send(b,d)*

*ack*  *b)*

*Fig. 4. Process senderRST S*

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

*PR = fsenderRSTSg*

*leading to the following integrated reactive state transformation system*

## *RST S = (DST S; P R)*

*Layer 4 (System Architecture of CCS Spec)*

*The integrated process signature P-SIG above leads to a basic component* COMP = ( ; P-SIG; P-SIG), where the semantics is given by the export P-SIG model RST S. In general the body of a basic component may include the construction of new processes by parallel composition of imported pro- cesses.

*4.2 Place/Transition Nets*

*Let PN = (P l; T r; pre; post; init) be a place/transition net with places P l,* transitions T r, pre- and post functions

*pre; post : Tr ! P l ;*

*where P l is the free commutative monoid over P l, and init 2 P l 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 di erent 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 SIG0 is the signature nat0 of natural numbers* SIG0 = nat0 = sorts : nat

*opns : zero ! nat*

*succ : nat ! nat*

*and the data value algebra is the algebra of natural numbers NAT = (IN; 0; +1):*

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

## *SIG = SIG0+*

*opns : p :! nat (p 2 P l)* with the following class of data states

*DS = fM = (NAT; (pM )p2Pl)jpM 2 IN g;*

*where each algebra M 2 DS with (pM )p2Pl can be considered as a marking* mM 2 P l de ned by

*mM = X pM p p2Pl*

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

*T = traf os : t : ; (t 2 T r);*

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

## *T-SIG = (SIG0; SIG;T ):*

*For each t 2 T that data state transformation tDSTS is given by*

*tDSTS = ftM1 ;M2 =M1; M2 2 DS and cond(tM1 ;M2 ) = trueg*

*where cond(tM1 ;M2 ) = true if*

*pre(t) mM1 and mM2 = mM1 pre(t)+ post(t):*

*In other words, cond(tM1 ;M2 ) = true means that transition t is activated in* marking mM1 and ring of t leads to marking mM2 : This leads to the following data state transition system

## *DST S = (A0; DS; T R)*

*with TR = (tDSTS)t2T and the data state transition system*

*GDSTS = (DS; T S; source; target)*

*with*

*TS = ftM1 ;M2 =M1; M2 2 DS; t 2 T r; cond(tM1 ;M2 ) = trueg source(tM1 ;M2 ) = M1 and target(tM1 ;M2 ) = M2:*

*This means that GDSTS 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 : ;*

*leading to the integrated process signature P-SIG = (T-SIG;P ). The corre-* sponding process netRSTS is given by

*netRSTS = (GRSTS(net); h(net));*

*where the reactive state transition system GRSTS(net) is the case graph of PN* (i.e. the subgraph of the marking graph reachable from the initial marking init) and h(net): GRSTS(net) ! GDSTS is the embedding of the case graph into the marking graph of PN. The integrated reactive state transition system RST S is given by

*RST S = (DST S; fnetRSTSg)*

*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) ! PN (i 2 I): The corresponding processes occ(i)RSTS are given by

*occ(i)RSTS = (GRSTS(occ(i)); h(occ(i)));*

*where the reactive state transition system GRSTS(occ(i)) is the case graph of* OCC(i) and h(occ(i)) : GRSTS(occ(i)) ! GDSTS is the morphism of the case graph into the marking graph GDSTS of PN induced by m(i) : OCC(i) ! PN . The integrated reactive state transition system RST S in this case is given by RST S = (DST S; P R) with

*PR = focc(i)RSTSji 2 Ig*

*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 I0 I, which represents a set of essential scenarios of the system. Especially I0 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 es-* pecially union and fusion. In the following we show how to represent the fusion of nets by a fusion component COMP3, which includes integrated pro- cess signatures P-SIG1 and P-SIG2 of two arbitrary Petri nets P N1 and P N2 represented by components COMP1 and COMP2. We compose these three components leading to a new component COMP4, which represents the fu- sion of P N1 and P N2:In more detail we have the following situation, where in the most basic case of fusion we have two Petri nets, P N1 and P N2 together with fusion places F1 of P N1, F2 of P N2, and a bijection f : F1 ! F2. The fusion P N3 of P N1 and P N2 via f , written P N3 = P N1 +f P N2, is the net P N3 obtained from the disjoint union of P N1 and P N2 by identi cation of corresponding fusion places. The given nets P N1 and P N2 can be considered for i = 1; 2 as basic components

*COMPi = (;; P-SIGi; P-SIGi);*

*where P-SIGi is the integrated process signature for P Ni as de ned in layer 3* above with semantics SEM(COMPi) given by the integrated reactive state transition system RST Si. The fusion construction of two Petri nets P N1 and P N2 discussed above can be represented by a basic component

## *COMP3 = (P-SIG1 + P-SIG2; BOD3; P-SIG3);*

*where P-SIG3 is the integrated process signature for P N3 = P N1 +f P N2, and*

## *BOD3 = (SIG0; SIG123; T123; P123)* SIG123 = SIG0 +

*opns : p :! nat (p 2 P l1 + P l2 + P l3)* T123 = traf os : t : ; (t 2 T r1 + T r2 + T r3)

*P123 = procs : net1 : ;*

*net2 : ; net3 : ;*

*By constructions we have natural embeddings from the disjoint union* P-SIG1 +P-SIG2(with shared SIG0) into BOD3 and from P-SIG3 into BOD3, where the construction of P123 corresponds to alternative 1 in layer 3. The semantics of COMP3 is intended to be a model transformation

*SEM(COMP3) : Mod(P-SIG1 + P-SIG2) ! Mod(P-SIG3);*

*which transforms the disjoint union RST S1 + RST S2 of integrated reactive* state transition systems RST S1 and RST S2 of Petri nets P N1 and P N2 cor- responding to P-SIG1 and P-SIG2 into the integrated reactive state transition system RST S3 of the fusion net P N3 = P N1 +f P N2. It remains open to give an explicit partial function on (P-SIG1 + P-SIG2)-models to realize this intended model transformation.

*In order to compose the components COMP1 and COMP2 with COMP3* we rst construct the disjoint union COMP1 + COMP2 (with shared SIG0), which has P-SIG1 +P-SIG2 in the export and in the body, and ; in the import. Now the disjoint union COMP1 + COMP2 can be composed with COMP3 leading to a (non-basic) component

## *COMP4 = COMP3 Æ (COMP1 + COMP2)*

*with empty import IMP4 and EXP4 = EXP3 = P-SIG3:* The strong composition COMP 0 is given in this case by

*4*

## *COMP 0 = (;; P-SIG3; BOD3)*

*4*

*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 speci cations 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 speci cation SP EC0, a SP EC0-algebra A, places P l, transitions T r, pre- and post- functions, and an eqns-function as- signing to each transition t 2 Tr sets pre(t); post(t); and eqns(t) respectively. As shown in gure 6, the arc-inscriptions of a high-level net N for a transition

|  |
| --- |
| *P-SIG 3* |
| *BOD3* |
| *P-SIG1 + P-SIG2* |

|  |
| --- |
| *BOD3* |
| *P-SIG 1 + P-SIG2* |
| *P-SIG1 + P-SIG2* |

|  |
| --- |
| *P-SIG 1* |
| *P-SIG1* |
|  |

|  |
| --- |
| *P-SIG 2* |
| *P-SIG 2* |
|  |

*Fig. 5. Composition of Petri Net Components*



*COMP3*

*BOD4*

*COMP*

*4*

*COMP1*

*COMP2*

*P-SIG 3*

*t 2 Tr wiht places p1; :::p*

*2 Pl in the pre-domain and p; ; :::p;*

*2 Pl in the*

*n 1 m*

*post-domain of t are given by multiset terms termi and termj with variables*

*over the signature SIG0 of SP EC0. The node-inscription of t is a set E of* equations over SIG0:



*p 1*

*p n*

*term1*

*termn*

*t*

*E*

*term’1*

*term’m*

*p’ 1*

*p’*

*m*

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

*In gure 6 the sets pre(t), post(t) and eqns(t) are given by* pre(t)= (term1; p1) :: (termn; pn);

*post(t)= (term; ; p; ) :: (term; ; p;*

*); and*

*1 1 m m*

*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 2 Pl a sort type(p) 2 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 2 Pl is an element d 2 As of the algebra A with s = type(p):

*Layer 1 (Data Types for AHL Nets)*

*The data value speci cation SP EC0 is the algebraic speci cation SP EC0 of* N and the data value algebra is the SP EC0 -algebra A of N.

*Layer 2 (Data States and Transformations for AHL Nets)*

*The data state speci cation SP EC is given by a multiset extension of SP EC0* with additional constant symbols p :! mult(type(p)) for each p 2 P l, where mult(type(p)) is the multisort of sort type(p). The class DS of data states is given by all those SP EC-algebras M, where the SP EC0-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 2 Tr we have exactly one transformation symbol t : v; , where v = s1::sn are the sorts si of the variables of the terms occurring in pre(t); post(t) and eqns(t).

*This leads to the following data state transformation speci cation*

## *T-SPEC = (SP EC0; SP EC; T );*

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

*DST S = (A; DS; TDSTS) and GDSTS = (DS; T S; source; target)*

*where GDSTS 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 GDSTS of N. In alternative 2 the processes are de ned as high-level processes of N, which are high-level occurrence nets OCC(i) with morphisms m(i) : OCC(i) ! 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 RST S would be morphisms from di erent initial markings into the marking graph GDSTS 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 dis- cuss 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 speci cations instead of signatures in layers 1 - 4 for the data type and data state parts.

*An attributed graph structure speci cation ATTR consists of a graph* structure signature, which is an algebraic signature with unary operation sym- bols only, an algebraic data type speci cation SP EC0 with sorts S0, and at- tribute assignment functions from each graph signature sort s to a suitable data type sort s0 2 S0.

*An attributed graph transformation system AGT according to [ELO95]* and [LKW93] consists of an attributed graph structure speci cation ATT R, a set of start graphs, and a set of transformation rules t : L ! R; where L and R are ATTR algebras: Given a rule t : L ! R and a match mL : L ! G an attributed graph transformation G ) H is de ned, 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 speci cation SP EC0 is the algebraic data type speci cation* of ATTR and the data value algebra is a given SP EC0-algebra A0.

*Layer 2 (Data States and Transformations for AGTs)*

*The data state speci cation SP EC is given by the attributed graph structure* speci cation AT T R of AGT: The class DS of data states is given by all AT T R- algebras A which are extensions of the data value algebra A0. For each rule t : L ! R of AGT we have a transformation symbol t : s1:::sn; , where s1:::sn 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 suÆcient to determine a match m : L ! G of the rule in G. Let T be the set of all these transformation symbols t : s1::sn; , then the data state transformation speci cation T-SPEC is given by

## *T-SPEC = (SP EC0; AT T R;T ):*

*The transformations of the AGT are de ning a data state transformation* system DST S and a data state transition system GDSTS, 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 de ned 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 de ned by speci c 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 con- struction 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 di er- ences 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 trans- formation 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 speci cations 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 speci cations instead of signatures becomes much more powerful, because constraints in the import can be considered as require- ments for the import while those in the export as properties of the component. This has been shown already for algebraic module speci cations in [EM90]. In [EO01] we discuss how essential parts of the theory of algebraic module speci cations in [EM90] can be extended to components for integrated data type and process speci cations. Several interesting concepts and examples for the speci cation 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 di erent kinds of Petri nets and graph transformations and a speci c CCS example in this paper. It remains open to study also in- stantiations for all the other examples of our conceptual model discussed in

*[EO98b,EO00], especially for di erent 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. Bussow, 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 Di erent Dimensions of Distributed System Speci cations. PhD thesis, Technische Universitat Berlin, 1998.*

*[DG94] P. Dauchy and M.C. Gaudel. Algebraic speci cations with implicit states.*

*Tech. Report, Univ. Paris Sud, 1994.*

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

*[EB94] H. Ehrig and R. Bardohl. Speci cation 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, Gro e-Rhode, and*

*J. Padberg. A new integration paradigm for formal speci cation of safe software systems. In Proc. 10th Japan-Germany Forum on Information Technology. Gesellschaft fur 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. Lowe, 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 Speci cation* *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 Speci cation 2: Module Speci cations 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 classi cation of data type and process speci cation techniques. Technical Report 98-10, Technical University of Berlin, 1998.*

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

*[EO00] H. Ehrig and F. Orejas. Integration and Classi cation of Data Type and Process Speci cation Techniques, 2000. submitted.*

*[EO01] H. Ehrig and F. Orejas. A Component Concept for Integrated Data Type and Process Speci cation Techniques. Technical report, Technische Universitat 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. Gro e{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. Gro e{Rhode. On a reference model for the formalization and integration of software speci cation languages. In Bulletin of the EATCS No 68, pages 81{89. 1999.*

*[Gro00] M. Gro e{Rhode. Compositional Comparison of Formal Software Speci cations using Transformation Systems, August 2000. submitted.*

*[Gro01] M. Gro e{Rhode. Semantic Integration of Heterogeneous Formal Speci cations 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. Classi cation 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 Scienti c, 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. Lowe, M. Kor , 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 Scienti c, 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 Scienti c, 1997.*

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

*[UML00] Uni ed Modeling Language { version 1.3, 2000. Available at* [*http://www.*](http://www/) *omg.org/uml.*