An executable Graphical Representation of Mediatory Information systems

Jacques Calmet * Dirk Debertin * Sebastian Jekutsch *
Joachim Schü *

Department of Computer Science
Institute for Algorithms and Cognitive Systems
University of Karlsruhe
{calmet,debertin,jekutsch,schue}@ira.uka.de

Abstract

In this paper we present an approach towards a unified modeling and query-processing tool for mediatory information systems. Based upon Coloured Petri-nets we are able to model the integration of parametric data (external, uncertain and temporal informations) and visualize the dataflow in mediatory information systems.

Keywords: Coloured Petri-nets, Interoperable federated systems, Parametric Databases, Deductive Databases

1 Introduction

With the transition from isolated to cooperating information systems there is an urgent need for sound computational query-processing with a diversity of circumstantial informations. Wiederhold et al. [27, 25, 26] have proposed the concept of a mediator, a device which expresses how the integration of different information systems is to be achieved. Different languages for building mediatory information systems have been presented in [17, 15, 12]. These approaches depart from former proposals as they are declarative, i.e. logic based. We believe that from a software engineering point of view a declarative language could alleviate the difficult task of building and in particular maintaining mediatory information systems. In this paper we present a Petri-net model based upon an extension of the Coloured Petri-nets [9] for such a language, which can be viewed as a parametric database [4] in the sense that it allows a seamless integration of temporal, uncertain and inconsistent information. The integration of external data is modeled as functions and relations over some external software packages. From an implementational point of view we use a common object request broker [2] to invoke the external functions and for describing the interface to those external functions/relations. For a detailed description on how external data can be integrated, the interested reader may refer to [17, 24]. We will focus in this paper on the graphical representation of the mediatory information system itself rather than the underlying information systems.

A step towards an environment for building mediated systems is a graphical user interface to assist the mediator developer. On the domain or object level numerous extensions of the entity-

*Author for Correspondence: Jacques Calmet. Department of Computer Science, Institute for Algorithms and Cognitive Systems, University of Karlsruhe, Am Fasanengarten 5, D-76128 Karlsruhe, Tel: +49 721 608 4208, Fax: +49 721 608 6116

Relation *STOCK*		
Name	*Date*	*Price*
BMW	17/5	771
Daimler	17/5	690
VW	17/5	410
BMW	16/5	771
Daimler	16/5	692
VW	16/5	414
BMW	15/5	770
Daimler	15/5	680
VW	15/5	405
BMW	14/5	766
Daimler	14/5	685
VW	14/5	406

Ticker 1 from 18/5, 11 am
`BMW 772 Daimler 685 VW 392`

Ticker 2 from 18/5, 11 am
`BMW 772 Daimler 684`

Table 1: External Information Systems in the Example

relationship model in order to model time, uncertainty and object-orientation have been proposed. Tools as [7, 23] for aiding partial or global schema integration provide assistance in figuring out possible equivalence among objects, but up to now no graphical tool modeling the dataflow in mediatory information systems has been made available.

Our model has several other applications: It provides a natural and simple mechanism for maintaining mediated views very different from algorithms such as [16]. Once the rules defining a view have been transformed into a Petri net, base relation updates and view redefinitions can be simply modeled by the addition or deletion of the corresponding places and transitions from the network. The well understood relationship of Petri nets and concurrency permits the simultaneous firing of several transitions. Another possible application of our model is the verification of large knowledge- and databases containing uncertain and temporal informations. The detection of redundancy, circularity and incompleteness by pattern recognition algorithms has been described in [28].

The paper is structured as follows. We first present the architecture of the mediatory information system followed by a description of a language for expressing mediatory knowledge and the coloured Petri-nets. In section 4, an extended Petri-net model for bottom-up evaluation of mediatory knowledge bases is presented. The paper is concluded by a comparison to former approaches, possible applications and further research directions.

2 Mediatory Architecture

The following example will be used throughout the paper to illustrate our approach. Consider a broker having access to a relational database containing stock prices of the last few days. Two different online tickers provide recent stock prices. These two differ in the amount of information they provide and in their reliability.

A frequent task of the broker is to discover extraordinary events, such as a stock differing more than 20 DM from its price in former times. He wants this task to be automated. The problems are that the two kinds of information sources – the relational DB and the online services – are heterogeneous in their structure, and that the two tickers may be inconsistent in the information they provide. See table 1 for example values.

Papakonstantinou et. al. provided in [21] an architecture for integrated access to heterogeneous information sources (IS). It consists of *mediators* converting queries in a common format (*OEM*) to more specialized queries, which in turn are converted by *translators* into queries in the language of the

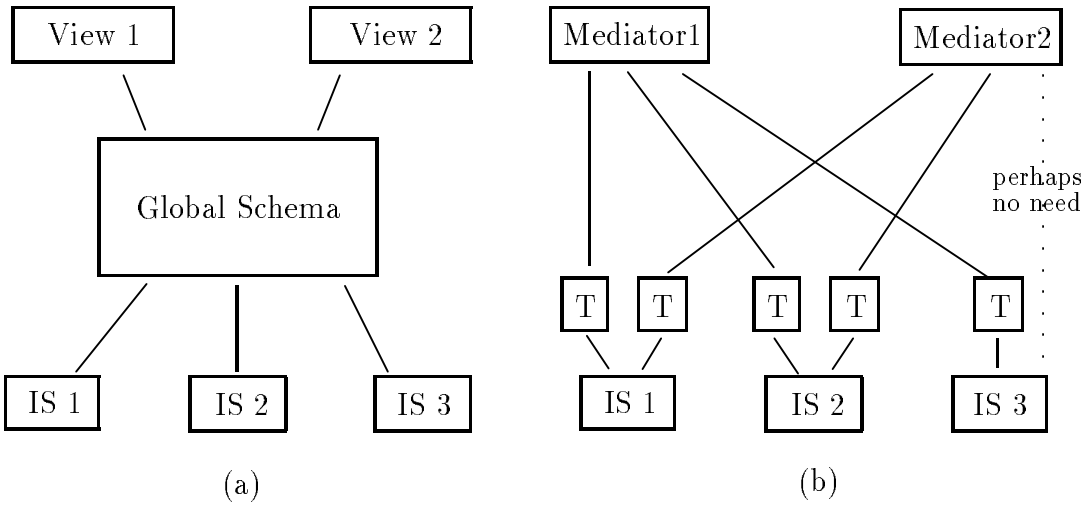


Figure 1: Top-Down-Integration of Heterogeneous Information Sources

requested IS. Such a translator exists for each mediator–IS combination. A translator also includes other functionalities for utilizing the IS for the mediator such as caching extracted information or managing remote procedure calls.

This approach differs significantly from traditional integration of multiple databases:

- There is no global schema integrating the local IS schemata.
- The information sources may be heterogeneous in their structure, for example object-oriented databases, computer algebra systems, flat WWW-pages or other mediators as well. Building global schemas require more or less uniform datamodels.
- Integrating heterogeneous information sources focus not only on the integration of the data-object schema but also on integrating the different accessing mechanisms. In fact this is the central point in [21].

Building a global schema from local schemas is a bottom-up procedure (see for example [22]): One explicitly defines the local schemas, constructs the global schema above them and finally defines application views onto the global schema (see Fig.1(a)). The mediatory methodology is in a more top-down fashion. First, there is a given application requirement for information access, for example having access to different stock information sources. The mediators providing this view have (via translators) *direct* access to the local information sources (IS), because – taken the view from the top – no global schema of all the IS is available (see Fig.1(b)). Therefore mediators can be seen as specialized "global" views. They only integrate those parts of the underlying IS that are crucial for the application needs they serve. Note that a mediator may be itself an IS for another mediator.

Our approach differs from [21] in that our mediator is knowledge based using a declarative rule based language. We believe that there is a need for AI-techniques in IS-integration for the following reasons:

- Two information systems may differ in the information they provide. Choosing the better (more reliable or more specialized) source requires expertise. In our example the broker has experience with the two tickers. He knows when to trust them. To simulate this behavior the mediator has to be some kind of expert system. Other knowledge intensive decisions may be

necessary such as choosing the optimal parameters for a query or consulting the sources in appropriate order.

- Given a predefined application need, it could be the case that neither each local information system nor a global schema build from them provides the whole answer. Instead, the answer to a query may require additional inferences on the external informations. In our example, none of the sources detect those extraordinary events.

These kinds of mediators do not only integrate heterogeneous IS but also add knowledge. The result is a new information system providing more than just the sum of the underlying information systems, but (re-)using and depending on them.

In this paper we will focus on the development of the mediator code. The construction of the translators, which make up an essential part of our architecture for mediatory knowledge bases will not be covered.

3 Prerequisites

3.1 A language for mediatory knowledge bases

The language for expressing mediatory knowledge bases has been proposed in [17, 24, 6, 14] and is based upon generalized annotated logic and constrained databases. In this section we first sketch generalized annotated logic introduced by M. Kifer and coworkers [13, 11]. It provides us with an universal language for dealing with temporal, uncertain and inconsistent information or in general with parametric data with provides the algebraic structure of a lattice. A description of the ongoing implementation for SUN and PC can be found in [6, 14]. For a comprehensive description of the language the reader may refer to [17, 11, 13].

3.1.1 Generalized Annotated Programs

Salient features of the language are the so-called *annotations* μ_i's which are constants, variables and terms over a complete lattice $\mathcal{T}$[1]. The following definitions are from [13]:

Definition 3.1 An *annotation* is either an element of $\mathcal{T}$ (c-annotation), an *annotation variable* (v-annotation) or a *complex annotation term* (t-annotation). Annotation terms are defined recursively as follows: members of $\mathcal{T}$ and variable annotations are annotation terms. In addition, if $x_1, \ldots, x_n$ are annotation terms, then $f(x_1, \ldots, x_n)$ is a complex annotation term.

If A is a usual atomic formula of datalog (in [13] predicate calculus) and μ is an annotation, then $A : \mu$ is an *annotated atom*. An annotated atom containing no occurrence of object variables is *ground*. A is called the *object part* and μ is called the *annotation part* of $A : \mu$.

Definition 3.2 (Annotated clause) If $A : \rho$ is an annotated atom and $B_1 : \mu_1, \ldots, B_k : \mu_k$ are c- or v- annotated atoms, then

$$A : \rho \leftarrow B_1 : \mu_1 \wedge \ldots \wedge B_k : \mu_k$$

is an *annotated clause*. $A : \rho$ is called the *head* of this clause, whereas $B_1 : \mu_1, \ldots, B_k : \mu_k$ is called the *body*. All variables (object or annotation) are implicitly universally quantified. Any set of annotated clauses is called a *Generalized Annotated Program (GAP)*.

[1] A complete lattice $(\mathcal{T}, \preceq)$ is a partial ordering with respect to $\preceq$, a least upper bound (lub) $\sqcup$ and a greatest lower bound (glb) $\sqcap$ for every subset of $\mathcal{T}$. A lattice is *linear* if $\preceq$ is a total ordering.

Definition 3.3 (Strictly ground instance) Suppose that C is an annotated clause. A *strictly ground instance* of C is any ground instance of C that contains only c-annotations.

Let H be the Herbrand base of the program. An annotated logic interpretation $\mathcal{I}$ is a mapping $\mathcal{I} : H \rightarrow \mathcal{T}$ from the base onto a lattice.

Definition 3.4 (Satisfiability) Let I be an interpretation, $\mu \in \mathcal{T}$ a c-annotation and A a ground atom:

1. $I \models A : \mu$ iff $I(A) \succeq \mu$.
2. $I \models \neg A : \mu$ iff $\neg(\mu) \preceq I(A)$.
3. $I \models F_1 \wedge F_2$ iff $I \models F_1$ and $I \models F_2$.
4. $I \models F_1 \vee F_2$ iff $I \models F_1$ or $I \models F_2$.
5. $I \models F_1 \leftarrow F_2$ iff $I \models F_1$ or $I \not\models F_2$.
6. $I \models F_1 \leftrightarrow F_2$ iff $I \models F_1 \leftarrow F_2$ and $I \models F_2 \leftarrow F_1$.
7. $I \models (\forall x)F$ iff $I \models \{x/t\}F$ for all ground terms t where x is an object-variable or annotation variable.
8. $I \models (\exists x)F$ iff $I \models \{x/t\}F$ for some ground term t where x is an object-variable or annotation variable.
9. If F is not a closed formula, then $I \models F$ iff $I \models (\forall)F$, where $(\forall)F$ denotes the universal closure of F.

There are two different kinds of negation in GAP, the so-called epistemic (or explicit) negation $\neg$ and the *non-monotonic* **not**. $\neg$ requires symmetry between **true** and **false**, e.g. $\neg A : \mathbf{t} = A : \mathbf{f}$. For a proposal on how to handle **not** by adapting the stable and well-founded semantics to GAP the reader may refer to [17]. We will not discuss this topic further, but the model presented here can easily be extended to deal with non-monotonic negation in stratified mediatory knowledge bases along the lines of [20]. For GAP without non-monotonic negation the fixed-point operator has the following form:

Definition 3.5 (Fixedpoint-operator) Let P be a generalized annotated logic program (GAP), $\mathcal{I}$ a GAP interpretation and $\mathcal{T}$ a complete lattice. Then a fixed-point operator $R_P(\mathcal{I})$ for bottom-up computation of GAP's is defined as follows: $R_P(\mathcal{I})(p) := \sqcup\{\rho \mid p : \rho \leftarrow p_1 : \mu_1, \ldots p_n : \mu_n$ is a strict ground instance of a clause in P and $\mathcal{I} \models p_1 : \mu_1, \ldots p_n : \mu_n\}$.

R_P may reach the least fixed point (lfp) if for all strict ground instances A, $lfp(R_P(A))$ is reached after a finite number of iterations. This condition, called *fixed-point reachability property* [13], holds for many GAP knowledge bases: If the clause bodies of a program contain only variable (v-) or only constant (c-) annotations, or if only finite or decreasing monotone functions[2] appear in the program. For instance, if the knowledge base consists of $Rains(Monday) : 0.5$ and $Rains(Monday) : 0.8$ the least upper bound computed by the fixed-point operator would be $Rains(Monday) : 0.8 = \sqcup\{0.5, 0.8\}$. Let $\mathcal{P}(\mathcal{M})$ denote the powerset of a given set M. Examples of useful lattices as truth domains are:

- Fuzzy-values: $([0,1], \leq)$ with the max function as lub.

[2] A function f is finite if $\{f(x)|x \in DOM(f)\}$ is finite and f is decreasing if for arbitrary arguments $x_1, \ldots x_n$ $f(x_1, \ldots x_n) \leq x_i$ for all $1 \leq i \leq n$.

- Inconsistent informations: The set FOUR [11] of $\{\mathbf{t}, \mathbf{f}, \top, \bot\}$ of truth values, where $\top$ stands for inconsistency and $\bot$ for "unknown". FOUR is a non-linear lattice, because $\mathbf{t}$ and $\mathbf{f}$ are not related.
- $Time_1$: The powerset of non-negative integers is a complete lattice when ordered under the $\subseteq$ ordering.
- $Time_2$: The set of all closed intervals of non-negative real numbers is a complete lattice when ordered under the $\subseteq$ ordering.
- Multiple Information Systems: The Information Systems (IS) identifier are stemming from the lattice $(\mathcal{P}(\{IS_1, \ldots, IS_n\}), \subseteq)$.

The expressiveness of the language is greatly improved by the *combination* of multiple lattices in two different ways. First the usual cross-product $\times$ with a componentwise ordering, least upper bound $\sqcup$ and greatest lower bound $\sqcap$. Second, the *free product*, denoted $\otimes$, which is similar to the cross product except that the lub is not defined pointwise but as a formal operation with the rules

$$\begin{aligned} (a \sqcup b) \otimes c &= (a \otimes c) \sqcup (b \otimes c) \\ a \otimes (b \sqcup c) &= (a \otimes b) \sqcup (a \otimes c) \end{aligned}$$

and additional rules which express commutativity and associativity of $\sqcup$ and distributivity of $\sqcup$ with $\sqcap$.

For instance the $[0,1] \times [0,1]$ cross-product has been useful in the area of probabilistic reasoning. The ordering is defined pointwise, therefore $[a_1, b_1] \preceq [a_2, b_2]$ iff $a_1 \leq a_2$ and $b_1 \leq b_2$. Consider as an example for the free-product point-based time and uncertainty. The database {Rains:[{Monday, Tuesday},0.5], Rains:[{Tuesday, Wednesday},0.6]} does not permit to answer the query Rains:[{Monday, Tuesday, Wednesday},0.6] since the least upper bound is not defined pointwise. Is is only possible to infer {Rains:[{Monday}, 0.5]} and {Rains:[{Tuesday, Wednesday},0.6]}. Intuitively Rains:[{Monday, Tuesday},0.5] means that it will rain on Monday and Tuesday with a certainty of at least 0.5.

Possibility of truth and falsehood	$N = [0,1] \times [0,1]$
Time and uncertainty	$T \otimes N$
Multiple information systems and uncertainty	$\mathcal{P}(1 \ldots n) \otimes N$
Time, multiple information systems and uncertainty	$T \otimes \mathcal{P}(1 \ldots n) \otimes N$

Table 2: Selected examples of (product)lattices

3.1.2 Integrating external Information Sources

The following description is taken from [16] and [17]. We add a constraint part to annotated clauses to enrich the expressiveness of GAPs and to provide a way to access external information systems.

Given a domain set $\mathcal{D}$ of objects that we wish to reason about, a set F of functions that are used to manipulate the objects and a set R of relations over F, a *constraint domain* $\Sigma_{\mathcal{D}}$ is defined as the tuple (F, R). A *constraint term* is build out of F in the obvious way. Note that an element $e \in \mathcal{D}$ may be viewed as a 0-ary function in F. An *atomic constraint* is a relation $r(a_1, \ldots a_n)$ where $r \in R$ and for all $1 \leq i \leq n$, a_i is a constraint term.

Access to an information system will be expressed as an atomic constraint. For example the relation $stock(Name, Date, Price)$ causes (via the translator) a SQL-call to the relational database containing the relation $STOCK$.

A *constraint* Ξ over $\Sigma_{\mathcal{D}}$ is a first order formula which is either true or false in $\Sigma_{\mathcal{D}}$. $\Sigma_{\mathcal{D}}$ is said to satisfy Ξ, denoted by $\Sigma_{\mathcal{D}} \rhd \Xi$, if

1. Ξ is an atomic constraint $r(a_1, \ldots a_n)$, then $(a_1 \ldots a_n) \in r$
2. Ξ is $\neg A$, then $\Sigma_{\mathcal{D}} \not\rhd A$
3. Ξ is $(A \wedge B)$, then $\Sigma_{\mathcal{D}} \rhd A$ and $\Sigma_{\mathcal{D}} \rhd B$
4. Ξ is $(A \vee B)$, then $\Sigma_{\mathcal{D}} \rhd A$ or $\Sigma_{\mathcal{D}} \rhd B$

A *constrained clause* C is an expression of the form

$$P_0 : \mu_0 \leftarrow \Xi \parallel P_1 : \mu_1 \wedge \ldots \wedge P_n : \mu_n$$

where the $\parallel$ is just a syntactical notion to separate ordinary GAP literals from the constraint part of a clause. If $\Sigma_{\mathcal{D}}$ is a constraint domain and $\mathcal{I}$ an interpretation then for the constrained clause C either $\Sigma_{\mathcal{D}} \rhd \neg\Xi$ or $\mathcal{I} \models P_1 : \mu_1 \wedge \ldots \wedge P_n : \mu_n$ holds.

Definition 3.6 (Mediatory Knowledge bases fixed-point-operator) Let P be a generalized annotated logic program (GAP) with constrained clauses, $\mathcal{I}$ a GAP interpretation , $\mathcal{T}$ a complete lattice and $\Sigma_{\mathcal{D}}$ a constraint domain. Then a fixed-point operator $T_P(\mathcal{I})$ for bottom-up computation of mediatory knowledge bases is defined as follows: $T_P(\mathcal{I})(p) := \sqcup\{\rho \mid p : \rho \leftarrow \Xi \parallel p_1 : \mu_1, \ldots p_n : \mu_n$ is a strict ground instance of a clause in P such that $\Sigma_{\mathcal{D}} \rhd \Xi$ and $\mathcal{I} \models p_1 : \mu_1, \ldots p_n : \mu_n\}$.

In the running example we need to integrate the information provided by the online tickers. We store this in the predicate $stock/2$ annotated by the source and date. *today* is a build-in constant. In our example it evaluates to 18/5. To invoke the tickers we use the following non-ground constrained facts:

$$stock(Name, Price) : [\{ticker_1\}, \{today\}] \leftarrow \underline{stock\_price1}(Name, Price) \parallel$$
$$stock(Name, Price) : [\{ticker_2\}, \{today\}] \leftarrow \underline{stock\_price2}(Name, Price) \parallel$$

The actual access to the external knowledge source is done when the $\underline{stock\_price}$ relations over the constraint domains $ticker_1$ and $ticker_2$ are evaluated.

The usefulness of this approach for mediated systems can be illustrated by the running example from section 2. The predicate

$$stock(Name, Price) : [\mathcal{P}(\{m, ticker1, ticker2\}), Date]$$

expresses the price of stock $Name \in \{bmw, daimler, vw, \ldots\}$ annotated with the source of the information – m being the mediator itself – and the date for which the price holds. In the following we will develop the example knowledge base.

Preference of information systems and incomplete knowledge. The variety of reasoning modes beyond local or global consistency applicable to mediation suggests that it may be better not to build a single mode of mediation but to provide the means to write clauses for mediation. In our example we need to express that the ticker 1 – although providing the additional information about the actual VW stock price – is not as reliable as ticker 2. Preferring ticker 2 can be expressed by the following two clauses:

$$\begin{aligned} stock(Name, Price_2) : [\{m\}, Date] \quad &\leftarrow \quad stock(Name, Price_1) : [\{ticker1\}, Date] \ \wedge \\ &\qquad\quad stock(Name, Price_2) : [\{ticker2\}, Date] \\ stock(Name, Price) : [\{m\}, Date] \quad &\leftarrow \quad stock(Name, Price) : [\{ticker1\}, Date] \ \wedge \\ &\qquad\quad \mathbf{not}(stock(Name, \_)) : [\{ticker2\}, Date] \end{aligned}$$

Given the example values in table 1 with this clauses we may conclude $stock(daimler, 684) : [\{m\}, \{18/5\}]$ as well as $stock(vw, 392) : [\{m\}, \{18/5\}]$.

Inferences about external information We would like to detect stocks whose price decreased more then 20 DM with respect to the days before.

$$selected\_stock(Name) : [\{m\}, \{today\}] \leftarrow Stock(Name, Date, Price) \wedge Price_1 \leq Price - 20 \parallel stock(Name, Price_1) : [\{m\}, \{today\}]$$

Note that when evaluating the *Stock* relation the external relational stock database is being accessed through some remote function call.

Query-processing with temporal data. In [13] it was shown how some kinds of temporal reasoning could be subsumed by GAP, e.g. clauses of the form

$$buy(Name) : succ(Date) \leftarrow selected\_stock(Name) : [\{m\}, Date]$$

where $succ()$ denotes a function which increases every member of the set $Date : 1995/10/0414 : 39 : 57$, e.g.
$succ(\{18/5\}) = \{19/5\}$.

Paraconsistent query-processing. Definite Horn-clauses used in ordinary deductive database languages lack the capability to express logical inconsistencies, intrinsic to a mediated environment. Using lattices like the above mentioned FOUR we are able to explicitly represent clauses which state how to deal with conflicting information, e.g. $buy(vw) : [\{m\}, \mathbf{f}\,] \leftarrow buy(vw) : [\{IS_1, IS_2\}, \top]$ states that whenever there is an attribute value conflict concerning the relation $buy(vw)$ in the information systems IS_1, IS_2 then the answer of the mediated system m should be false. As one can see now the computation of the least upper bound in the above fixed-point operator is an essential operation for sound query-processing in the mediatory knowledge base, otherwise we would not be able to answer queries like $buy(vw) : [\{IS_1, IS_2\}, \top]$ with respect to a knowledge base containing $buy(vw) : [\{IS_1\}, \mathbf{t}]$, $buy(vw) : [\{IS_2\}, \mathbf{f}\,]$.

3.2 Coloured Petri-nets

A Coloured Petri-net is a triple $N = (P, T, A)$ consisting of disjoint sets P (*places*) and T (*transitions*) and a multi-set A (*arcs*) over $(P \times T) \cup (T \times P)$ forming a bipartite graph. Each place $p \in P$ is assigned a *colourset* $C(p)$ and a multi-set $M(p)$ of *tokens* each of colour $C(p)$. Coloursets can be viewed as datatypes, and tokens are instances having a specific colour. To each arc $a = \langle p, t\rangle \in A$ or $\langle t, p\rangle \in A$ is attached a *label* $L(a)$ of type $C(p)$. Note that tokens as well as labels may contain variables of suitable type. A *marking* is the distribution of tokens over all places of the net. Each transition $t \in T$ is assigned a *boolean guard* $G(t)$ expressing constraints on the variables binded to t. See [9] for an extended and more formal definition of coloured petri-nets.

Let $IN(t) := \{\langle p, t\rangle \in A | p \in P\}$, $OUT(t) := \{\langle p, t\rangle \in A | p \in p\}$, $\bullet t := \{p \,|\langle p, t\rangle \in A\}$ and $t\bullet := \{p \,|\langle t, p\rangle \in A\}$ denote the vicinity of $t \in T$. A transition t is called *enabled* iff the following conditions hold:

- For each incoming arc $a_i \in IN(t)$ there is at least one variable substitution σ_i such that a token $s \in M(p)$ exists with $\sigma_i(s) = \sigma_i(L(a_i))$. This particular token s must not serve again as a resource for another substitution σ_j for $j \neq i$. Recall that $M(p)$ is a multi-set, therefore more than one token of this kind may be present.
- All substitutions σ_i are compatible. σ_i and σ_j are compatible if their concatenation $\sigma_i\sigma_j$ is defined. In other words there is no assignment of two different values to the same variable.

- If $G(t)$ evaluates to true under $\sigma = \sigma_1\sigma_2\cdots\sigma_m$ then σ is called an *enabling substitution*.

There could be more than one enabling substitution σ under the same marking. A transition could *fire* if it is enabled under a substitution σ. If a transition t fires, the tokens M_i of the places are updated to M_{i+1} as follows:

$$M_{i+1}(p) := \begin{cases} M_i(p) \setminus \sigma(L(\langle p,t\rangle)) & : \ p \in {\bullet}t \setminus t{\bullet} \\ M_i(p) \cup \sigma(L(\langle t,p\rangle)) & : \ p \in t{\bullet} \setminus {\bullet}t \\ M_i(p) \setminus \sigma(L(\langle p,t\rangle)) \cup \sigma(L(\langle t,p\rangle)) & : \ p \in {\bullet}t \cap t{\bullet} \\ M_i(p) & : \ \text{otherwise} \end{cases}$$

Given a marking M_0, a sequence $t_1, \ldots t_n$ is called a *firing sequence* if for each i $(1 \leq i \leq n)$ t_i is enabled under the marking M_{i-1} and t_i's firing results in the marking M_i. The firing sequence changes the marking M_0 into M_n.

4 An extended Petri-net Model

In this section the representation of mediatory knowledge and the integration of external information sources via remote function calls is described.

A transformation of a GAP knowledge base into an extended Petri-net $N = (P, T, A)$ is done according to the subsequent rules (the clauses are enumerated from 1 to n):

- Each predicate p is a place p in the net.
- Each clause c is a transition c $(1 \leq c \leq n)$ in the net.
- Let O be the type of the object part and $\mathcal{T}$ the annotated lattice of predicate p. Then $C(p) := O \times \mathcal{T}$.
- For every clause c of the form

  $$p_0(o_0) : \mu_0 \leftarrow p_1(o_1) : \mu_1 \wedge \ldots \wedge p_{m_c}(o_{m_c}) : \mu_{m_c}$$

  and $1 \leq i \leq m_c$, the net contains the arcs $a_i := \langle p_i, c\rangle$ with the labels $L(a_i) := (o_i, \bar{\mu_i})$ where $\bar{\mu_i}$ is a new variable annotation. $\mu_i \preceq \bar{\mu_i}$ is added as conjunctional condition to the guard of transition c if μ_i is a c-annotation. In addition the net contains the arc $a_0 := \langle c, p_0\rangle$ with the label $L(a_0) := (o_0, \bar{\mu_0})$ where

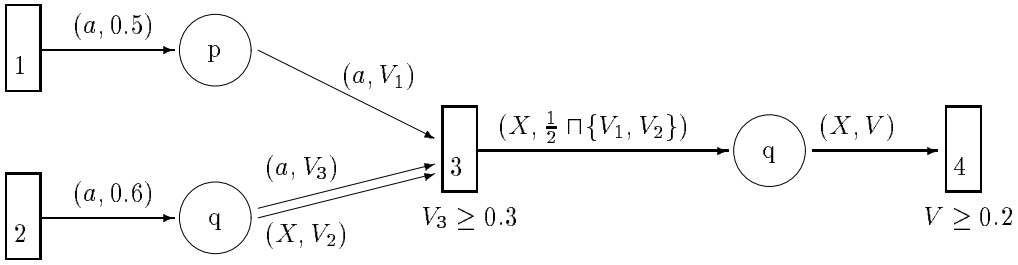
  $$\bar{\mu_0} := \begin{cases} \mu_0 & : \ \mu_0 \text{ is c-annotation} \\ \sqcap\{\bar{\mu_i} | \mu_i \text{ is the same variable as } \mu_0\} & : \ \mu_0 \text{ is v-annotation} \\ f(\bar{\rho_1}, \ldots \bar{\rho_n}) & : \ \mu_0 = f(\rho_1, \ldots \rho_n) \end{cases}$$

  The $\bar{\rho_1}, \ldots \bar{\rho_n}$ are defined recursively similarly. Note that $\sqcap\{a\} = a$ for every $a \in \mathcal{T}$ and $\sqcap\{\} := \sqcap\mathcal{T}$.
- The initial marking is empty, that is $\forall p \in P : M_0(p) = \emptyset$.

Queries can be added to the net as they are headless clauses. The following abstract example illustrates the transformation in its details. Places are drawn as circles and transitions as rectangles. We omit typing information and assume that $C(p) = C(q) = C(r) = \{a\} \times [0, 1]$. All uppercase letters are variables.

Example 1

(1) $p(a) : 0.5 \leftarrow$
(2) $q(a) : 0.6 \leftarrow$
(3) $r(X) : \frac{1}{2}V \leftarrow p(a) : V \ \wedge \ q(X) : V \ \wedge \ q(a) : 0.3$
(4) $\leftarrow r(X) : 0.2$



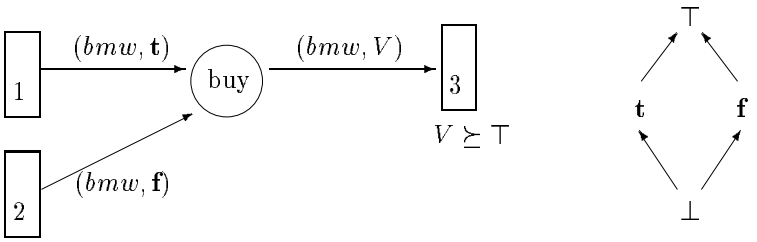
In the following a substitution is written as a set of bindings of the form X/t where X is a variable and t is a term of appropriate colour. In example 1 answering to the query $r(X) : V$ can be modeled by a firing sequence $1, 2, 3, 4$: Transitions 1 and 2 are always enabled since their guards are true and no variable binding is necessary. Their firings place the tokens $(a, 0.5)$ in p and $(a, 0.6)$ in q. Consider now transition 3. A possible binding is $\sigma = \{V_1/0.5, V_2/0.6, V_3/0.6, X/a\}$. Due to the fact that the guard $V_3 \geq 0.3$ evaluates to true under σ, transition 3 is enabled. Its firing (see below for problems here) adds the token $\sigma((X, 0.5 \cdot \sqcap\{V_1, V_2\})) = (a, 0.25)$ to place r. Finally the query transition 4 is enabled with $\sigma = \{X/a, V/0.25\}$ which is also the binding for the successful query.

We need to extend the model in the following three ways in order to capture the fixed-point semantics 3.6 of a mediatory knowledge base:

1. In the example above only one token was in place q after transition 2 fired, but transition 3 needed this token *two* times to be enabled, one for every arc $\langle \mathrm{q}, 3 \rangle$. Unlike the definition in section 3.2 in our model tokens will *not* be removed if a transition fires. This reflects that the tokens represent knowledge rather than resources that cannot be shared. It is similar to the case when a materialized view gets queried, where an element from a view should not be deleted if it has been queried for. In other words our Petri-net model caches all facts necessary for answering a query which could lead to a large number of tokens to be kept within the net. Such an extension avoids conflicts between transitions which need the same token to be enabled, as encountered in the example.

2. The model presented so far only works with linear annotation lattices. Consider the following example using the non-linear lattice FOUR. The ordering of the elements in FOUR is illustrated at the right of the picture.

Example 2



After the firings of 1 and 2 *buy* contains the tokens $(bmw, \mathbf{t})$ and $(bmw, \mathbf{f})$. There are two possible substitutions for V: $\{V/\mathbf{t}\}$ and $\{V/\mathbf{f}\}$. None of them satisfies the guard $V \succeq \top$ hence transition 3 is not enabled. This is a contradiction to the fixed-point semantics of GAPs, because $\sqcup\{\mathbf{t}, \mathbf{f}\} = \top$. In the example a token $(bmw, \top)$ should be in $M(p)$ although none of the incoming transitions 1 and 2 delivered it. We call such derived tokens *reductants* [13][3].

Definition 4.1 (Reductants) Given a set $M = \{(o_1, \mu_1), \ldots (o_n, \mu_n)\}$ of tokens and a unifier σ with $\sigma(o_1) = \ldots = \sigma(o_n)$ the token $(\sigma(o_1), \sqcup\{\mu_1, \ldots \mu_n\})$ is called a *reductant*. The function *reductants(M)* computes the set of reductants derived from all subsets of M for which σ is defined.

For example $((a, b), \top)$ is a reductant of the set $\{((\mathrm{X},\mathrm{b}),\mathbf{t}), ((\mathrm{a},\mathrm{Y}),\mathbf{f})\}$. It is important that every annotation in M is a c-annotation to ensure that the least upper bound $\sqcup$ is defined. For markings $M(p)$ this is always the case according to the next theorem. A proof has appeared in [3].

3. It is also possible to delete tokens from a place. For example every time $(a, 0.5) \in M(p)$ serves as a token for an enabling substitution for a transition t (with $\langle p, t\rangle \in A$), $(a, 0.6)$ will as well; but not vice versa. We say that $(a, 0.6)$ *subsumes* $(a, 0.5)$, because $0.6 \geq 0.5$. $(a, 0.5)$ might be deleted from $M(p)$ without changing the behavior of an extended Petri-net.

Theorem 1 (Possible tokens of a place) Let P be a GAP and N its transformation. At all places $p \in P$ of $N = (T, P, A)$ there are only tokens $(o, \mu) \in M(p)$ with $\mu \in \mathcal{T}$ if P is finite.

Definition 4.2 (Subsumption) Given two tokens $(o_1, \mu_1), (o_2, \mu_2) \in M$, the first *subsumes* the second if $\mu_1 \succeq \mu_2$ and there exists a substitution σ such that $o_2 = \sigma(o_1)$. The function *subsumption(M)* computes all tokens in M which are subsumed by at least one other token in M.

For example $(a, \mathbf{t})$ and $(a, \mathbf{f})$ are both subsumed by their reductant $(a, \top)$, whereas $subsumption(\{((X, b), \mathbf{t}), ((a, Y), \mathbf{f}), ((a, b), \top)\})$ is empty.

To summarize the three extensions presented above we redefine the update of the marking due to the firing of transition $t \in T$:

$$
\begin{array}{lrcl}
(1) & M_{i+1}^{up}(p) & := & \begin{cases} M_i(p) \cup \sigma(L(\langle t, p\rangle)) & : \quad p \in t\bullet \\ M_i(p) & : \quad otherwise \end{cases} \\
(2) & M_{i+1}^{red}(p) & := & M_{i+1}^{up}(p) \cup reductants(M_{i+1}^{up}(p)) \\
(3) & M_{i+1}(p) & := & M_{i+1}^{red}(p) \setminus subsumption(M_{i+1}^{red}(p))
\end{array}
$$

With this extension example 2 works as expected: Transitions 1 and 2 place the tokens $(bmw, \mathbf{t})$ and $(bmw, \mathbf{f})$ in p respectively. $M^{red}(p)$ evaluates to $\{(bmw, \mathbf{t}), (bmw, \mathbf{f}), (bmw, \top)\}$ and $M(p)$ to $\{(bmw, \top)\}$, which enables transition 3, since $\top \succeq \top$.

Before presenting algorithms for the testing for firability of a transition and updating of the net marking, some more definitions are required:

[3] Different from the definition provided here, in [13] derived *rules* are named reductants. Note that tokens are representations for annotated atoms due to the presented transformation.

Definition 4.3 (Unification $mgu_{\mathbf{a}}()$ of tokens) Tokens $s = (o, \mu)$ as well as arc labels consist of two parts, its first being the object part $s^{obj} = o$ and the second being the annotation part $s^{ann} = \mu$. Let $mgu(o_1, o_2)$ denotes the usual most general unification of o_1 and o_2. Given two tokens/labels s_1, s_2 the *most general annotational unifier* $mgu_{\mathbf{a}}(s_1, s_2)$ is constructed as follows:

$$mgu_{\mathbf{a}}(s_1, s_2) := \begin{cases} mgu(s_1^{obj}, s_2^{obj}) \cup \{s_2^{ann}/s_1^{ann}\} & : \ s_2^{ann} \text{ is v-annotation} \\ mgu(s_1^{obj}, s_2^{obj}) & : \ s_2^{ann} \text{ is c-annot. and } s_1^{ann} \succeq s_2^{ann} \\ undefined & : \ \text{otherwise} \end{cases}$$

If $mgu(s_1^{obj}, s_2^{obj})$ is not defined, $mgu_{\mathbf{a}}(s_1, s_2)$ is not defined either. Note that $mgu_{\mathbf{a}}()$ is not symmetric. If $mgu_{\mathbf{a}}(s_1, s_2)$ is defined, s_1 and s_2 are said to be *unifiable*.

For example, $mgu_{\mathbf{a}}(((X, a), 0.5), ((b, Y), 0.4)) = \{X/b, Y/a\}$ and $mgu_{\mathbf{a}}((a, \mathbf{t}), (a, \mathbf{f}))$ is not defined.

Definition 4.4 (Compatibility of annotation substitutions) Two substitutions $\{V/a\}$ and $\{V/b\}$ which assign different c-annotations a and b to the same annotation variable V are *compatible* if a and b are comparable due to the ordering of the underling lattice. In this case their *concatenation* $\{V/a\}\{V/b\}$ is defined as $\{V/ \sqcap \{a, b\}\}$. This definition is easily extended to cases with more than two substitution.

Definition 4.5 (Concatenation of $mgu_{\mathbf{a}}$s) The substitutions in $mgu_{\mathbf{a}}$s may be divided in object variable substitutions and annotation variable substitutions. The *concatenation* $\sigma_1\sigma_2\cdots\sigma_n$ of n $mgu_{\mathbf{a}}$s $\sigma_1, \ldots \sigma_n$ is defined as the usual concatenation of the object variable substitutions unioned with the above defined concatenation of the annotation variable substitutions.

Testing for fireability

Input: Extended Petri-net $N = (P, T, A)$;
Transition $c \in T$
Output: Maximal set of $mgu_{\mathbf{a}}$s each enabling c. c is not enabled if the set is empty.

$\Theta := \{\}$; *(Θ is a set of sets of possible substitutions for c)*
for all arcs $a \in IN(c)$ **do**
(Let a be $(p, c) \in A$)
$\varphi_a := \{\}$; *(φ_a is the set of all possible substitutions for a)*
for all tokens $s \in M(p)$ **do**
if $unifiable(s, L(a))$ **then** $\varphi_a := \varphi_a \cup mgu_{\mathbf{a}}(s, L(a))$;
if $\varphi_a = \{\}$ **then return** $\{\}$;
$\Theta := \Theta \cup \varphi_a$;
(Let Θ be $\{\varphi_1, \ldots \varphi_{|\Theta|}\}$)
$\Psi := \{\}$; *(Ψ is a set of all enabling substitutions for c)*
for all permutations $(\sigma_1, \ldots \sigma_{|\Theta|})$ with $\sigma_i \in \varphi_i \in \Theta\ (1 \leq i \leq |\Theta|)$ **do**
if $\sigma_1, \ldots \sigma_{|\Theta|}$ are pairwise compatible **then** $\Psi := \Psi \cup \sigma_1\sigma_2\cdots\sigma_{|\Theta|}$;
return Ψ;

Firing of a transition

Input: Extended Petri-net $N = (P, T, A)$;
Transition $c \in T$;
Set Ψ of c-enabling substitutions
Output: N with updated marking using every $\sigma \in \Psi$

for all arcs $a \in OUT(c)$ **do**
(Let a be $(c,p) \in A$)
for all $\sigma \in \Psi$ **do**
$M(p) := M(p) \cup \{\sigma(L(a))\}$;
$M(p) := M(p) \cup reductants(M(p))$;
$M(p) := M(p) \setminus subsumption(M(p))$;

See [14] for algorithms implementing $reductants()$ and $subsumption()$.

It is worth noting that our model captures the operational semantics of a GAP, which means that if there is a GAP for which the least fixed-point reachability property does not hold (e.g. from $\{p:0, p:\frac{1+x}{2} \leftarrow p:x, q:1 \leftarrow p:1\}$ it is never possible to answer the query $q:1$) the corresponding Petri-net cannot answer this query as well and runs forever.

So far only the transformation of GAPs to the extended Petri-net model has been described. The constraint part of a clause c, which is essential for the integration of external information sources, will simply be added to the guard $G(c)$of transition c. For c to be enabled, $G(c)$ – including the constraints – has to be satisfied. This implements the fixpoint operator T_P. The following theorems have been proven in [3] and capture the soundness and completeness of the proposed extended Petri-net model with respect to the semantics of constrained GAP:

Theorem 2 (Soundness) Let P be a GAP with clauses $c_1, \ldots c_n$, c_n a query and N the extended Petri-net defined on P. If there is a successful firing sequence in N then $c_1, \ldots c_{n-1} \models c_n$.

Theorem 3 (Completeness) Let P be a GAP with clauses $c_1, \ldots c_n$, c_n a query and N the extended Petri-net defined on P. If $c_1, \ldots c_{n-1} \models c_n$, then there is a successful firing sequence in N.

Figure 2 (refer to [21] for a similar picture) illustrates the mediatory architecture with the extended Petri-net. It refers to the broker example and shows the clause which computes the selected stocks. For the purpose of readability the annotations are being omitted. The transition is enabled with the substitution $\{Name/vw, Price_1/392\}$. The relational database is invoked to return convenient tuples. Only one tuple returned by the translator is drawn. This tuple – $(414, 16/5)$ – passes the guard so place *selected_stock* will be updated as shown.

5 Conclusion and further research directions

In this paper we presented a Petri-net model as an executable graphical representation of mediating knowledge.

In [19] a Petri net model for reasoning in the presence of inconsistency has been presented. It is based upon a preliminary version of the logic we are using in this paper and does not allow query-processing with temporal and uncertain informations. In particular v- and t- annotations have not been addressed, which are necessary for dealing with temporal/uncertain information and aggregation. However, their Petri-net models first-order GAPs. Meseguer [18] developed a consistency checking algorithm for a propositional rule base based upon Predicate/Transition Petri nets [5]. Similar to the work of Bell et al. [1] for computing materialized views by mixed-integer programming methods, his approach defines the notion of consistency through the solution of a system of linear equations obtained from the Petri net representation. However, our use of the Coloured Petri net departs from the Place/transitions nets sufficient for propositional logic. In [28] a tool for knowledge verification has been presented in which the task of knowledge validation becomes the identification of certain subgraph isomorphisms of the Petri-net. Inconsistent, redundant, subsumed, circular and

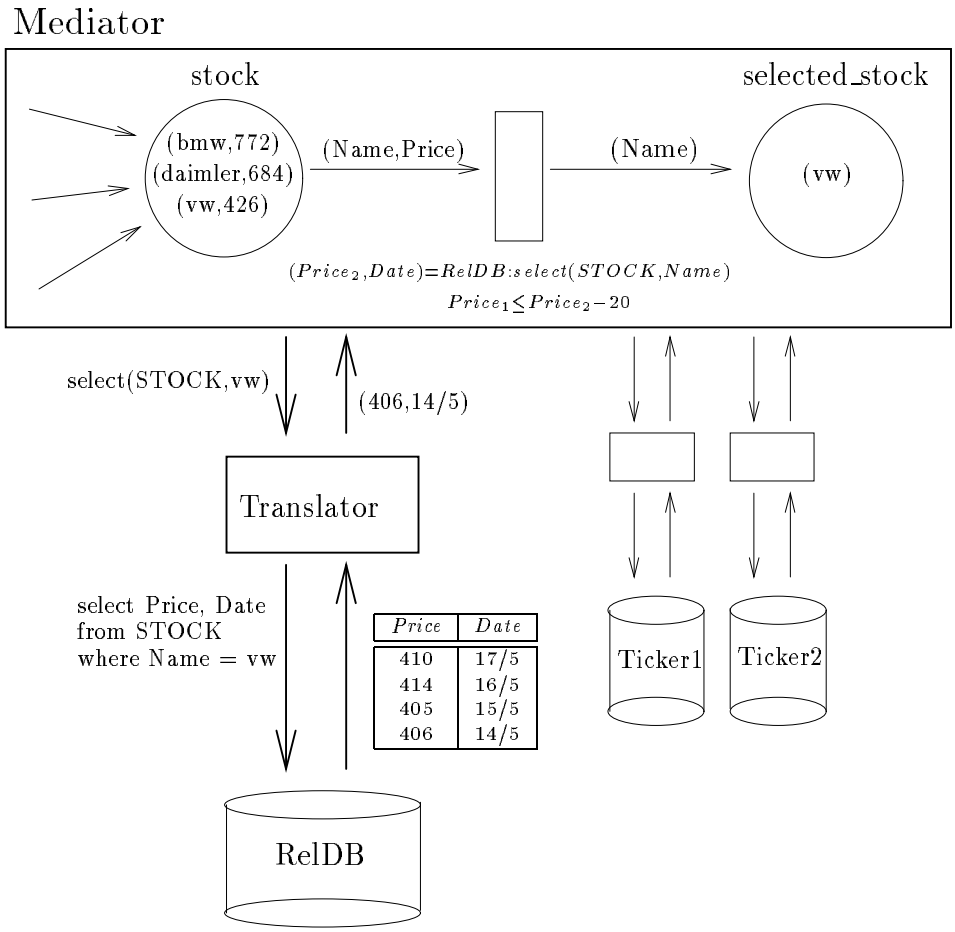


Figure 2: Mediatory knowledge base in action

incomplete rules are defined as patterns of the Petri net model and detected by a syntactic pattern recognition method. We believe that their method can also be applied using our model description but only for detecting redundant rules and incompleteness which means to identify those rules which can never fire or do not have any impact on the rest of the net. With respect to classical relational languages we explicitly allow to specify how to deal with conflicting information, instead of an expensive detection of possibly contradicting rules. Therefore detection of conflicting rules becomes unnecessary in our language.

The algorithms presented are currently under implementation [8]. The tool provides a graphical user-interface for entering and executing GAPs. A goal is to provide a methodology supported by a tool for top-down-construction of mediator code. We believe that a graphical representation could not only facilitate the acquisition of mediating rules but also provides some hints about a possible distribution of the mediatory knowledge base across a network. However the model only specifies the data flow but not the control flow. From a software engineering point of view its a good point to separate them, but a control flow specification – that is prescribing the order in which enabled transitions fire – is necessary to provide efficient execution.

We did not discuss the development of translators in this paper. A translator depends on the information system it is put on and therefore different translators are necessary for different

sources. We believe that at least parts of a translator will be reusable for various information sources, so building a translator shell is one of our future goals. Furthermore we plan a detailed performance analysis of the Petri-net approach for incremental view maintenance in comparison to other algorithms such as [16].

References

[1] Colin Bell, Anil Nerode, Raymond Ng and V.S. Subrahmanian. Implementing Deductive Databases by Linear Programming. *Proceedings of ACM Symposium on Principles of Database Systems, 1992, pp. 283-292*

[2] ORBIX, distributed object technology. IONA Technologies Ltd. ORBIX- A Technical Overview, July, 1993

[3] D. Debertin. Parallizing inference in distributed mediated systems. *Master's thesis, Department of Computer Science, IAKS, University of Karlsruhe (in German)*

[4] Sashi K. Gadia. Parametric databases: seamless integration of spatial, temporal, belief and ordinary data. *SIGMOD Record, Vol.22, No.1, March 1993, pp.15-20*

[5] H.J. Genrich. Predicate/Transition nets. *LNCS 254, Springer-Verlag, 1987 pp. 207-247*

[6] V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, Jim Lu, Adil Rajput, T.J. Rogers, R. Ross, C.Ward. HERMES Heterogenous Reasoning and Mediator System. Draft, University of Maryland, 1995 (available through WWW).

[7] Stephen Hayne, Sudha Ram. Multi-User View Integration System (MUVIS) : An Expert for View Integration. *IEEE Conference on Data Engineering, 1990, pp.402-409*

[8] Sebastian Jekutsch. A graphical Tool for Construction And Debugging of GAPs. *Project work, Department of Computer Science, IAKS, University of Karlsruhe (in german)*

[9] Kurt Jensen. Coloured Petri Nets: A High Level Language for System Design and Analysis. in: G. Rozenberg (Ed.): Advances in Petri Nets 1990

[10] P. Kanellakis, G. Kuper and P. Revesz. Constraint Query Languages. *ACM Symposium on Principles of Database Systems, pp. 299-313, 1990*

[11] Michael Kifer, E. Loziniskii. RI: a logic for reasoning with inconsistency. *Journal of Automated Reasoning 9, 1992, pp.179-215*

[12] R. Krishnamurthy, W. Litwin and W. Kent. Language features for interoperability of databases with schematic discrepancies. *Proceedings ACM SIGMOD, 1991, pp. 40-49*

[13] Michael Kifer, V.S. Subrahmanian. Theory of Generalized Annotated Logic. *Journal of Logic Programming 12, 1992, pp. 335-367*

[14] Peter Kullmann. SLG-Resolution for Generalized Annotated Logic (in preperation). *Master's thesis, Institute for Algorithms and Cognitive Systems, University of Karlsruhe, 1995* (in German)

[15] A. Lefebvre, P. Bernus, R. Topor. Query Transformation for Accessing Heterogenous Databases. *Joint International Conference and Symposium on Logic Programming, Workshop on Deductive Databases, 1992*

[16] Jim Lu, Guido Moerkotte, Joachim Schü, V.S. Subrahmanian. Efficient Maintenance of Materialized Mediated Views. Accepted for ACM SIGMOD, 1995

[17] Jim Lu, Anil Nerode, V.S. Subrahmanian. Towards a Theory of Hybrid Knowledge Bases. *To appear in IEEE Transactions on Knowledge and Data Engineering*

[18] P. Meseguer. A new method to checking rule bases for inconsistency: a Petri Net approach *Proceedings of ECAI, Stockholm, 1990, pp. 437-442*

[19] Tadao Murata, V.S. Subrahmanian, Toshiro Wakayama. A Petri Net Model for Reasoning in the Presence of Inconsistency. *IEEE Transactions on Knowledge and Data engineering, Vol3, No.3, September 1991, pp.281-292*

[20] T. Shimura, J. Lobo, Tadao Murata. A Petri Net Semantics for Logic Programs with Negation. *Proceedings of the 1992 International Conference on Software Engineering and Knowledge Engineering, Capri, Italy, pp.292-299*

[21] Yannis Papakonstantinou, Hector Garcia-Molina, Jennifer Widom. Object Exchange Across Heterogenous Information Sources. Department of Computer Science, Stanford University. Available via WWW.

[22] M.P.Reddy, B.E.Prasad, P.G.Reddy, Amar Gupta. A methodology for integration of heterogenous databases. *IEEE Transactions on Knowledge and Data Engineering, Vol.6, No.6, December 1994*

[23] Amit P. Sheth, James A. Larson. A Tool for Integrating Conceptual Schemas and User Views. *IEEE Conference on Data Engineering, 1988, pp.176-183*

[24] V.S. Subrahmanian. Amalgamating Knowledge Bases. *ACM Transactions on Database Systems 19,2, 1994, pp. 291-331*

[25] Gio Wiederhold. Intelligent Integration of Information. *ACM SIGMOD Conference, 1993, pp. 434-437*

[26] Gio Wiederhold. Interoperation, Mediation, and Ontologies. Available via WWW

[27] Gio Wiederhold, Sushil Jajodia and Witold Litwin. Integrating temporal data in a heterogenous environment. *In: Temporal Databases. Benjamin/Cummings, Jan 1993*

[28] D. Zhang, D. Nguyen. PREPARE: A Tool for Knowledge Base Verification. *IEEE Transactions on Knowledge and Data Engineering, 1994, Vol. 6, Number 6, pp. 983-989*