Validating Access to External Information Sources in a Mediator Environment (Technical Report)

Zoltán Ádám MANN
Budapest University of Technology and Economics
Department of Control Engineering and Information Technology
zoltan.mann@cs.bme.hu

Supervisors:
Jacques CALMET, Peter KULLMANN
University of Karlsruhe
Institute for Algorithms und Cognitive Systems
{calmet,kullmann}@ira.uka.de

September 15, 2001

Abstract

A mediator integrates existing information sources into a new application. In order to answer complex queries, the mediator splits them up into sub-queries which it sends to the information sources. Afterwards, it combines the replies to answer the original query. Since the information sources are usually external, autonomous systems, the access to them can sometimes be erroneous, most notably when the information source is changed. This results in an incorrect behaviour of the whole system. The question that this paper addresses, is: how to check whether or not the access was correct?

The paper introduces a notational framework for the general information access validation problem, describes the typical errors that can occur in a mediator environment, and proposes several validation mechanisms. It is also investigated how the validation functionality can be integrated into the mediator architecture, and what the most important quality measures of a validation method are. Moreover, the practical usability of the presented approaches is demonstrated on a real-world application using Web-based information sources. Several measurements are performed to compare the presented methods with previous work in the field.

Key words and phrases: mediator, validation, wrapper verification, information integration

1 Introduction and previous work

In the past decades a tremendous amount of data has been stored in electronic form. In recent years, as a consequence of the unbelievable evolution of the World Wide Web, practically any information one can imagine is to be found in some format or the other on the WWW.

However, this is not enough for the future information society. The problem is not the amount of available information, which is already more than sufficient, but its usability. Information sources (ISs) are designed for their particular purposes, but need to be reused in completely different applications, in conjunction with other pieces of information. This not only holds for the Web but also for a variety of other ISs.

As an example, consider the development of a decision support system (DSS [23]). This may require the integration of ISs such as various relational and object-oriented databases, electronic documents, information from SAP[1], Web-based ISs, documents in EDI[2] format, computer algebra systems or special software libraries.

1.1 Mediators

To address the problem of integrating heterogeneous, autonomous ISs, Wiederhold suggested the *mediator* pattern in [28]. The mediator implements the common tasks of splitting complex queries into simpler ones that can be sent to the underlying ISs and combining the replies to an answer to the original query. The latter also includes the detection and handling of potential conflicts that may arise if two ISs return contradictory results.

The ISs are bound into the mediator architecture via *wrapper*s: components that translate queries from the language of the mediator into that of the ISs and the answers from the language of the IS into that of the mediator. The resulting architecture is depicted in the UML diagram of figure 1. More information on mediators can be found *e.g.* in [27, 12, 29].

The context of the work presented in this paper was provided by KOMET (Karlsruhe Open MEdiator Technology [4]), a logic-based mediator shell developed at the University of Karlsruhe. KOMET uses the declarative language KAMEL (KArlsruhe MEdiator Language), which is based on annotated logic. It provides a framework for the easy construction of mediators, and enables the reuse of existing code. It also supports conjunctive queries and negations. The external ISs are defined as predicates, and the queries are processed by an inference engine.

A particular mediator system, called MetaSearch [5], which is implemented using KOMET, was of special interest. MetaSearch is a meta Web search program that takes queries and delegates them to various Internet search engines such as AltaVista[3] and Google[4]. It then combines the answers of the search engines into one answer page.

MetaSearch is very important for two reasons. First, with the spread of the World Wide Web, the integration of Web-based ISs becomes a major challenge and the most important application for mediator systems, and MetaSearch can be regarded as a prototype

[1] SAP is a wide-spread enterprise information and control system
[2] Electronic Data Interchange [9]
[3] `http://www.altavista.com`
[4] `http://www.google.com`

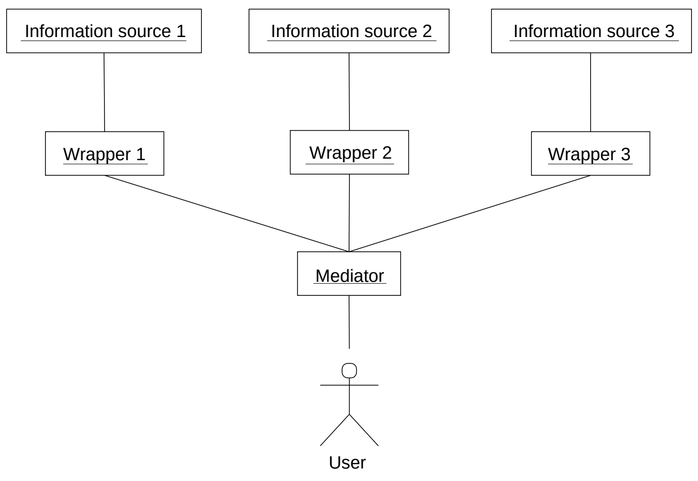

Figure 1: The mediator architecture

of such applications. Second, although MetaSearch is a relatively simple mediator application, it demonstrates very well the features of KOMET and also the problems that arise as a consequence of integrating external, autonomous ISs.

Since the actual access to the external ISs is performed in the wrappers, they are of special importance from this paper's point of view. In the case of MetaSearch, the conversion from the language of the mediator into that of the IS is simple: it boils down to encoding the query into a URL. For instance, to send the query 'Titanic' to AltaVista, the URL `http://www.altavista.com/cgi_bin/query?q=Titanic` has to be fetched. The conversion of the answer from the language of the IS into that of the mediator is somewhat more challenging, since the actual answer – the URL, title and excerpt of about 10 relevant pages – is embedded into an HTML page, along with plenty of other data, such as banners and statistics. Therefore the task of the wrapper is to extract the actual answer from the resulting page. In MetaSearch, regular expressions are used for that purpose. For instance in the case of AltaVista, the expression `<dl><dt><b>*.</b><a href="%U"><b>%T</b></a><dd>` could be used to extract the URL and the title, where the URL is stored in the variable `%U`, and the title in `%T`.

1.2 Validation

Although regular expressions are not the only possibility for the extraction of information from HTML (and also it is not the objective of this paper to study such mechanisms extensively), this method illustrates the problem well: the information extraction mechanism has to rely on some regularity of the output of the IS. Since the IS is an autonomous system, it may be altered, which may in turn cause the wrapper to stop extracting correctly.

There has already been some research on this problem, though not much. The most relevant work is that of Kushmerick [16, 18]. His algorithm RAPTURE uses statistical features, such as length, number of words, number of special characters *etc.* to characterize the extracted text segments. It learns the parameters of normal distributions describing the feature distributions of the extracted pieces of texts. These normal distributions are used to estimate the probability that a new wrapper output is correct – taking one particular feature into account. These probabilities are then combined to estimate the overall probability that the new wrapper output is correct.

Much of the remaining scientific work focuses on the automatic creation of wrappers (see *e.g.* [10, 11, 17, 22] and references therein), but there are also some results that can be used for the wrapper validation problem as well. Cohen [6] uses a notion of textual similarity to find "structure" in Web pages: this is useful mainly in wrapper induction but may also be used in wrapper validation and maintenance, because it can detect changes in the structure of the HTML page. Lerman *et. al.* [20] developed DATAPRO, an algorithm that uses tokens (words or generalizations of words) to represent text, and learns significant sequences of tokens – sequences that are encountered significantly more often than would be expected by chance. Thus the tuples to be extracted can be characterized by a set of significant token sequences, and a change in them can help uncover changes in the HTML layout.

It can be seen that although there are already some promising results, the problem of access validation needs more thorough research because the existing results apply only to a very restricted problem class. The most important restrictions are:

- only Web-based applications have been investigated;
- only errors produced by layout changes have been covered;
- only syntactic validation methods for the wrapper output have been proposed.

Accordingly, this paper addresses the more general question: how to check whether or not the access to an external IS was correct? One of the main contributions of the paper is a notational framework that enables definition of the general information access validation problem. Also, existing results can be placed naturally in this framework.

Another major problem with the existing results is that although the proposed validation methods can detect almost every change in the HTML layout, they often give false positives, *i.e.* they relatively often claim correct wrapper outputs to be erroneous. This problem is also studied in depth.

1.3 Paper organization

Section 2 first formalizes the information access validation problem (IAVP) in its most general form and defines the most important sub-problems, giving a notational framework for the rest of the paper. It also enables prooving the hardness of the IAVP. Section 3 gives an overview on the typical errors that can occur during the access to external ISs. Section 4 presents several quality measures of the validation process and demonstrates why the high rate of false positives is an intrinsic property of access validation. Then in section 5, several validation methods are presented that can be used generally for access validation in a mediator environment. In particular, the applicability in a Web-based setting is also covered. Section 6 investigates how the validation functionality can be integrated into the mediator architecture. Section 7 is a case study: it illustrates how the presented methods can be applied in practice, namely in the case of MetaSearch. It is demonstrated with several measurements, how – with appropriate techniques – better validation quality can be achieved. Section 8 ends the paper with a conclusion.

2 Problem definition

This section provides a general notational framework for the investigation of the information access validation problem (IAVP).

Definition 1 (Messages on communication channels) *Let the alphabet of the communication system be the finite set Σ. Thus, messages are elements of Σ^*. ε, the empty string, is also an element of Σ^*. Let ν be a symbol with $\nu \notin \Sigma^*$. ν means 'no message'.*

The following definition describes the interface of ISs. Note that – since ISs are assumed to be completely autonomous – we have no knowledge about their inner state. The only thing we know is that an IS accepts queries, to which it (usually) gives a reply. (See also figure 2.)

Definition 2 (Information source (IS)) *An information source is a family of functions $I_t : \Sigma^* \to \Sigma^* \cup \nu$. ($t \in \mathbb{R}$.) $I_t(q) = r, r \in \Sigma^*$ means that, at time t, the IS's reply on query q is r. On the other hand, if $r = \nu$, this means that the IS gave no reply.*

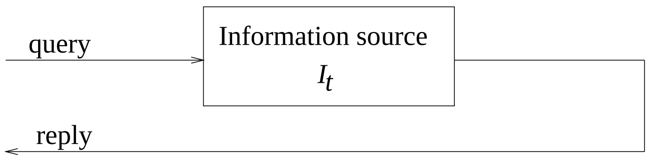

Figure 2: Model of the IS

Definition 3 (Wrapper) *A wrapper W is a pair of Turing machines, $W = (T, E)$, where T is a translator and E is an extractor. The translator implements a function $F_T : \Sigma^* \to \Sigma^*$, translating queries. The extractor implements a function $F_E : \Sigma^* \cup \nu \to \Sigma^*$, extracting the replies of ISs.*

Note that, in contrast to ISs, wrappers have a known inner structure (Turing machine). Moreover, they do not change over time. It is also important to note that even if the IS gives no reply, the wrapper has to return something, *i.e.* $F_E(\nu) \in \Sigma^*$.

Now the task is to implement the validator, *i.e.* the function that can tell if the access was correct or not:

Definition 4 (Validator) *A validator is a function $V : \Sigma^* \times (\Sigma^* \cup \nu) \times \Sigma^* \times S_W \times H \to \{correct, incorrect\}$. $V(q, r, r', s_W, h)$ is the validator's judgement on the* *wrapper output r', extracted from the IS's reply r, given on query q; s_W is any state information of wrapper W, and h is any additional 'historical' information that the validator has collected over time.*

It can be seen from this definition that the validator may make use of many observations and different kinds of knowledge; however, it does not have to. As noted in the Introduction, previous work has focused on validators using only the wrapper output r'.

Note that the above definition allows many different validators; the objective is to create the best validator in some sense:

Definition 5 (The general information access validation problem (IAVP)) *Let U be the set of possible validators, $f : U \to \mathbb{R}$ the objective function. The general IAVP consists of constructing the validator V^* which maximizes f on U.*

It would be logical to use the objective function

$$f_0(V) = \begin{cases} 1 & \text{if } V'\text{s judgement is always right} \\ 0 & \text{otherwise} \end{cases}$$

which would mean finding the *perfect* validator. However, creating such a validator is infeasible:

Remark 6 *The perfect validator has to be able to tell if the IS or the wrapper has fallen into an infinite loop. Therefore, the perfect validator would solve the Halting Problem of Turing machines, which is not algorithmically solvable.*

Consequently, we will have to settle for a less ambitious objective. That is, we are looking for methods that work well in most practical cases. To that end, section 3 reviews the typical errors that will have to be coped with, and section 4 presents more realistic objective functions for the evaluation of validators.

Before that, one more definition is given, because a restricted set of ISs, namely search engines, will be of special importance later on in the paper.

Definition 7 (search engine) *A search engine is an IS with $I_t(q) = (T, U, X)^c$, where c is the number of tuples returned, and each tuple consists of the title T, the URL U, and the excerpt X of a Web page.*

3 Typical errors

In order to solve the IAVP, it has to be clarified first what kind of errors are to be reckoned with during the access to external ISs. In this section, the typical errors are classified in three different aspects: according to the source of the error, the symptom caused by the error, and the duration of the error.

3.1 Classification of error sources

The most obvious error sources are the wrapper, the IS and the communication channel between them. However, an erroneous behaviour can also be the consequence of a *change* in the IS, causing incompatibility between wrapper and IS. Similar incompatibilities could theoretically also be caused by changes of the wrapper or the communication channel; however, such a problem has not been reported so far.

The continuous change in content and layout is especially present in Web-based ISs. Kushmerick [16, 18] investigated 27 actual sites for a period of 6 months, and found that 44% of the sites changed its layout during that period at least once. Changes in the IS can further be classified – according to the definition that ISs implement a family of function $r = I_t(q)$ – as follows:

- changes in the query language of the IS
- changes in the underlying set of information itself
- changes in the presentation of the replies

These error sources are investigated in the following in more detail.

3.1.1 Errors in the IS

No system is perfectly reliable, and the ISs are no exceptions either. They are autonomous systems, which makes the detection of such errors a tough problem, and their prevention or correction is practically impossible since the ISs are beyond our control. Also, typically no precise wisdom about the reliability of the used IS is available, and it can also degrade over time.

A typical example in the Internet is that the server that the IS resides on is – transiently or permanently – shut down. This will usually cause the wrapper to extract nothing. However, an empty reply does not necessarily indicate such an error since the IS may reply with the empty string for some queries. This makes such errors relatively hard to cope with. (Using the notations of section 2, the problem is then often $F_E^{-1}(\varepsilon)$ has more than one element, typically at least ν and ε.)

Unfortunately, there are many even subtler errors than this. For instance:

- bugs in a used software package can cause unpredictable results, ranging from erroneous replies to infinite loops;
- the IS may itself contain contradictory information;
- the IS may contain information that is not correct.

As can be seen, some of these errors can be handled easily in the wrapper itself (*e.g.* that the IS accepts no queries), but others – most notably semantic errors – are really hard to discover (*e.g.* that the IS contains corrupt information). The latter would require a lot more knowledge and intelligence.

The same can be observed in the case of MetaSearch: it is easy to recognize that the search engine is not responding. But it is not so easily decided if the information contained in the recommended pages is actually relevant to the query.

3.1.2 Errors in the communication channel

The wrapper usually cannot access the IS directly, but through some communication channel. The kind of errors that can occur at this point depends heavily on the type of the communication channel. Of course, the least error-prone situation is when the IS resides locally. But problems can also occur in such a situation, *e.g.* buffer overflow, deadlock *etc.*, as a consequence of insufficient resources or bad operating system design.

Matters become worse if the access takes place over the network. Naturally, the kind of used the network service makes a big difference. Most systems would be better off with a (perhaps virtual) private network with guaranteed QoS than the Internet. However, as already noted, the Internet is one of the most important application platforms for mediator systems.

Besides simple technical errors – like a power outage –, there are many other more complicated errors, above the physical layer. Typical examples include DNS errors, IP resolution conflicts, packet loss, proxy failures and the migration of sites.

From the wrapper's point of view it is often hard to tell (and sometimes it does not really matter, either) if a particular error occurred in the IS or in the communication infrastructure. For instance, if a Web page cannot be retrieved, this may mean that the page has been removed, the server is shut down, or there is no route to the server because of a proxy error.

3.1.3 Errors in the wrapper

Wrappers are implemented either manually or automatically. The resulting wrapper can be incorrect in both cases.

If the wrapper was created manually, software errors (bugs as well as design flaws) have to be taken into account. It is worth noting that the testing of wrappers – as a piece of software – is similar to the IAVP (see section 2) and accordingly extremely difficult. The wrapper has to cope with an infinite number of possible inputs, generated by two autonomous entities: the user on the one hand and the IS on the other.

There are some promising results on wrapper induction, *i.e.* the automatic generation of wrappers (see e.g. [11, 17, 22, 25]). This minimizes the risk of errors introduced by the 'human factor'; however, none of the suggested learning algorithms is perfect. Even if the learning algorithm provably converges to the correct wrapper, the result of a finite number of steps is usually imperfect. For instance, a wrapper could learn, if taught with too few training examples, that each URL starts with `http://`, which can of course cause problems in the case of an URL starting with `ftp://`. This problem is rendered even harder by the fact that the learning algorithms have to be trained in most cases with positive examples only [15, 20].

3.1.4 Changes in the underlying information base

Small changes of the information base are tolerated in most cases unproblematically by the wrapper and thus by the mediator system. For instance, if an Internet search engine adds new pages to its database, or deletes old, obsolete pages, this should not make any difference in its use or usability: such changes usually remain unobserved.

Big changes on the other hand can affect usability heavily. For instance, if the IS becomes specialized so that it can only answer queries of a special kind, this will usually cause problems in the mediator system.

However, even small changes play an important role in access validation. It is because of these small, quotidian changes that the testing of wrappers is very difficult, because it cannot be taken for granted that the IS answers the same query q always with the same reply r, or in other words: that I_t depends on t. Hence, standard test algorithms such as regression testing cannot be applied directly to the IAVP [16].

3.1.5 Changes in the query language of the IS

Although the query language of the ISs seldom changes (with respect to the frequency of changes in the content of the IS or the layout of the replies), it is not impossible. For instance, such a change was observed in the case of AltaVista on May 1, 2001, during our case study (see section 7 for details).

Usually, such changes are easily detected, because the wrapper will most probably either extract nothing or something that is obviously not the answer. For example, if the input handling of a search engine is modified (*e.g.* it accepts the queries in a different variable, or even through a different script), it is likely to return an error message if used in the old setting. The error message is a HTML page describing the circumstances of the error. Naturally, the wrapper will not be able to extract anything sensible from such a page.

Theoretically, more subtle errors of this type can occur too (*e.g.* changes that affect certain queries only). However, such changes are quite rare, and hence their importance is lower.

3.1.6 Changes in the format of the replies

As already mentioned, changes in the presentation of the replies is the most frequent error source, especially in the case of Web-based ISs. Web sites tend to change their layout often, which can have a number of reasons, for example:

- the embedded ads are changed or new ads are added
- the set of supported features is increased (*e.g.* a new possibility is created to enable searching MP3-files only)
- the user interface is simplified or some flaws in the user interface design are corrected

Whether a change in the layout of the replies results in an actual error, depends of course heavily on the wrapper. Wrappers should be constructed in such a way that they can tolerate the small and unimportant changes (which are also the most frequent ones), *e.g.* the changes in the embedded ads. On the other hand, wrappers must make use of some formatting regularities of the layout of the replies. Hence, a complete redesign of the layout – which also happens now and then – will prohibit the wrapper from working correctly.

3.2 Classification of symptoms

After having enumerated the various error sources, now an overview of the possible symptoms is given.

In the preceding paragraphs it was often mentioned what symptom a given error is likely to produce. However, in practice, usually the opposite is needed: facing some symptom, it has to be decided what went wrong (or, if anything went wrong at all).

3.2.1 Problems while connecting

Many errors of the underlying technical infrastructure (network, proxy, server, DNS-server *etc.*) become visible already when connecting to the external IS. This is advantageous, because this makes it clear that the problem is in the technical infrastructure and not in the actual IS. This information is vital, because otherwise – if, for example the output of the IS would be treated as an empty page in such a situation – it could not be decided later if an error occurred or the reply of the IS is really ε ($\in \Sigma^*$). What follows is that extra care has to be taken in the wrapper construction process to realize precise error handling in the connection phase.

3.2.2 The wrapper falls into an infinite loop

This symptom seems to be rather rare. However, it is theoretically possible, and it cannot be recognized algorithmically (see section 2). In practice, however, a time-out value can be specified with the property that the wrapper is most probably in an infinite loop if its net running time is above the time-out value.

The cause for such a symptom is most probably a bug in the code of the wrapper. It is possible though that this bug is activated by a change in the IS, which makes an implicit assumption of the programmer invalid.

3.2.3 The wrapper extracts nothing

The most frequent symptom is that the wrapper extracts nothing. As already mentioned in section 3.1, several types of errors can cause this symptom. What is more, it can also happen without any error, if the reply of the IS was ε. So this symptom does not necessarily imply an error.

In order to differentiate between these situations, test queries can be used: queries that will obviously produce a reply other that ε. More on this in section 5.

3.2.4 The wrapper extracts an incorrect number of tuples

This symptom is a generalization of the previous one. The reason why the special case was discussed separately is that it occurs more frequently than any other instance of the general phenomenon.

This symptom is useful if the IS uses a relational data model AND it is known (at least approximately) how many tuples the IS will return. For instance, AltaVista almost always returns 10 tuples, and it surely never returns more than 10 tuples. So if the wrapper extracts more than 10 tuples, that will most probably be the evidence of an error.

If the data model of the IS is not relational, then the length of the returned information can be used for similar validation purposes.

3.2.5 The extracted information is syntactically incorrect

If there was no problem while connecting to the remote IS, the wrapper did not fall in an infinite loop, and it returned a plausible number of tuples, then the last chance to filter out many errors without much additional knowledge is a syntactical analysis of the returned information.

As an example of a syntactically incorrect reply, consider the following situation. Querying the names of countries with a given property, the tuples of the reply are like „`><tr><td><img src=`“.

Kushmerick found [18] that most wrappers extracting from HTML give similar results when the layout of the reply of the IS has changed. Hence, many errors can be detected using methods working on the syntax level.

3.2.6 The extracted information is semantically incorrect

The most tough errors are those that produce false but syntactically correct replies, *i.e.* replies that seem to be correct. In order to detect such errors, additional knowledge is necessary. The problem is that in order to detect *all* such errors, the validator must have at least the same amount of knowledge that is expected from the IS. This is of course infeasible because if this knowledge was available locally, there would be no need to use external ISs.

On the other hand, there are semantic errors that can be detected with less additional knowledge. Continuing the already mentioned example, suppose that the names of those countries are queried where the currency is called ’dollar’, and the reply of the IS – as extracted by the wrapper – is ’Sadarfeguk’. This is syntactically correct because this string might be the name of a country where the currency is called dollar. However, it is not, and the obvious reason is that there is no such country at all. In order to check this, only a list of existing countries is needed (or maybe an external IS from which it can be queried if a country with a given name exists). Thus, in order to detect this particular semantic symptom, less additional knowledge was needed than that of the IS, because no information about the currencies of the countries was used by the validator.

At this point, it can be argued that this is a task for the mediator. However, this is a validation task, which can be integrated indeed into the mediator, but not necessarily. See section 6 for a discussion on how to integrate validation functionality into the mediator architecture.

3.3 Classification according to the duration of the error

Finally, it is important to differentiate between transient and permanent errors, because this aspect has large impact on the way the error should be fixed.

3.3.1 Transient errors and changes

Many problems of the underlying technical infrastructure are transient. For instance, if a server is not responding because it is too busy, repeating the query may solve the problem. In the case of a more severe problem, *e.g.* if the proxy breaks down, it could take hours or even days to fix the problem. In any case, it is not necessary to modify the mediator system in any way. The transient error might not even affect the work of the mediator system and thus remain unobserved.

3.3.2 Permanent errors and changes

In the case of a permanent error or change, the mediator system must usually be adapted to the new situation. This might involve reprogramming certain parts of the software, or an automatic repair (if, for instance, the wrapper was constructed automatically, it has to be reconstructed).

A typical permanent error is that an IS ceases to exist. In this case, the mediator system has to be changed so that it uses other ISs – if that is possible. A typical permanent change is that the layout of the replies of an IS is changed. In this case the corresponding wrapper must also be adapted.

4 Quality measures of validation

As already stated in section 2, it is infeasible to strive after a perfect validator. Rather, the aim should be to construct a validator that is as 'good' as possible. This section tries to formalize the word 'good'. The benefit of this is threefold:

1. the presented quality measures can guide the construction of high-quality validators;
2. a well-defined comparison of the validation mechanisms (to be presented in section 5) is made possible;
3. the introduced formalism sheds some light on common problems with the validation methods presented in the literature, and it can be proven why this is intrinsic.

4.1 Functional quality measures

If it is infeasible to expect that the validator's judgement always be correct, it is a natural requirement that its correctness should be as high as possible, in a statistical sense. In order to formalize this, assume that the validator performs N tests, *i.e.* it validates N accesses of a wrapper W to an external IS I_t. The number of flawless accesses is k, the number of accesses during which an error occurred is $N-k$. In each test, the validator must give a judgement whether or not the access was erroneous. The number of tests in which the judgement of the validator is right is denoted by m, thus the number of tests in which the judgement of the validator is false, is $N-m$.

Definition 8 (statistical correctness (SC)) *The statistical correctness of the validator is*

$$SC = \frac{m}{N}$$

Note that the SC (also called accuracy) is a number in the [0,1] interval, and that higher values are better. Also note that this number does not depend on the value of k. But, what does this number express? For instance, is a SC of 0.9 bad or good? Is a SC of 0.999 much better than that of 0.99? The answer of these questions does depend on the value of k, or more precisely on the value of k/N.

Generally, it can be assumed that the wrapper works properly most of the time, and errors rarely occur. This implies $k \approx N$. This is important because in the case of a 'dull' validator that simply judges every access to be correct, $m = k$ and thus the statistical correctness is

$$SC_{\text{dull}} = \frac{k}{N} \approx 1!$$

As can be seen, even a very high value of C can be bad if it is not significantly higher than k/N. What follows is that the SC alone is not expressive enough. As can be seen, the cause for this phenomenon is that one of the outcomes of the test has a much higher probability than the other.

A similar problem arises in the medical sciences, in the context of routine tests for diseases. The doctor uses some test to check if the patient has a particular disease, *e.g.* AIDS. Since only a small minority of the population is infected, the probability that the patient suffers from the given disease is very low. The problem of evaluating the quality of a particular test methodology is analogous to the problem of evaluating validators. To solve this problem, they use different quality measures that are more expressive than SC.

In order to define these quality measures, we need to distinguish between primary and secondary errors (see table 1). Note that these are the possible errors that the validator can make (and not the wrapper this time!).

	The access was actually	
Judgement of the validator	correct	erroneous
access was correct	OK	secondary error
access was erroneous	primary error	OK

Table 1: Primary vs. secondary error

In the formalism of statistics, the validator's task is hypothesis testing. The null hypothesis (H_0) is that the wrapper functions properly and the access was correct. The alternative hypothesis is that the access was erroneous. Its judgement is the result of the hypothesis test [8].

Based on this picture, the following quality measures are defined in the medical sciences [7]:

Definition 9 (specifity and sensitivity)

$$\begin{aligned} \textit{specifity} &= Pr(\textit{test result is negative}|\textit{patient is healthy}) \\ \textit{sensitivity} &= Pr(\textit{test result is positive}|\textit{patient is ill}) \end{aligned}$$

Adapted to the case of validation:

$$\begin{aligned} \textit{specifity} &= Pr(\textit{validator's judgement is}: \textit{'correct'}|\textit{access was correct}) \\ \textit{sensitivity} &= Pr(\textit{validator's judgement is}: \textit{'erroneous'}|\textit{access was erroneous}) \end{aligned}$$

Remark 10 *These values can be regarded as the two components of the SC, since*

$$\begin{aligned} &SC = Pr(\textit{judgement is true}) = Pr(\textit{judgement is true}, \textit{access was correct}) + \\ &+Pr(\textit{judgement is true}, \textit{access was erroneous}) = \\ &= \textit{specifity} \cdot Pr(\textit{access was correct}) + \textit{sensitivity} \cdot Pr(\textit{access was erroneous}) \end{aligned}$$

The definition of specifity and sensitivity is symmetric, but the large difference between the probabilities of the possible judgements makes them differently expressive. In order to clarify this, we introduce the reverse conditional probabilities:

Definition 11 (positive and negative predictive values)

$$\begin{aligned} \textit{negative predictive value} &= Pr(\textit{access correct}|\textit{judgement is}: \textit{'correct'}) \\ \textit{positive predictive value} &= Pr(\textit{access erroneous}|\textit{judgement is}: \textit{'erroneous'}) \end{aligned}$$

These are actually the most important quality measures because they show how 'trustworthy' the validator is. That is: if the validator judges the access to be correct/erroneous, what is the probability that the access was really correct/erroneous, respectively?

All four values (specifity, sensitivity and the predictive values) are in the [0,1] interval, and higher values are better. There is an interesting connection between these values, as captured by the theorem below. But first some abbreviations are introduced:

$$\begin{aligned} x &= \text{specifity} \\ y &= \text{sensitivity} \\ p &= Pr(\text{the access was erroneous}) \\ q &= Pr(\text{judgement is}: \text{'erroneous'}) \\ PPV &= \text{positive predictive value} \\ NPV &= \text{negative predictive value} \end{aligned}$$

Theorem 12 *(i) If $x \to 1$ and $y \to 1$ and $p \to 0$ and $q \to 0$, then $NPV \to 1$, but PPV does not necessarily converge to 1.*
(ii) If $x \to 1$ and $y \to 1$, then

$$\frac{\partial PPV}{\partial x} \to \frac{1-p}{p} \quad \textit{and} \quad \frac{\partial PPV}{\partial y} \to 0.$$

Before we go on to prove the theorem, first its actual meaning and significance should be clarified:

Remark 13 *As already noted, usually the wrapper functions correctly. It can also be assumed that the judgement of the validator is correct in most cases. This is why the theorem deals only with the case when x and y are high (near 1) and p and q are low*

(near 0). In this ideal case, one would expect that both predictive values also converge to 1. The first claim of the theorem shows that this is true for NPV, but not for PPV! If PPV is not high enough, this means that the validator gives many false positives. This is exactly the reason why the validation mechanisms presented in the literature suffer from a relatively large number of false positives. This is an intrinsic property of the IAVP. So, an otherwise almost perfect validator guarantees high NPV but not necessarily a high PPV.
The second claim of the theorem explains the reason of this phenomenon. It turns out that in the region where both specifity and sensitivity are high, PPV does not depend on the sensitivity anymore, but it depends heavily on the specifity. Note that $\frac{1-p}{p}$ is a huge number. Consequently, if the specifity is a little bit under 1, the positive predictive value suffers from it severely.

Proof. Using Bayes' theorem,

$$\begin{aligned} NPV &= \frac{Pr(\text{access correct, judgement : 'correct'})}{Pr(\text{judgement : 'correct'})} = \\ &= \frac{\text{specifity} \cdot Pr(\text{access correct})}{Pr(\text{judgement : 'correct'})} = \frac{x(1-p)}{1-q} \to 1 \end{aligned}$$

$$\begin{aligned} PPV &= \frac{Pr(\text{access erroneous, judgement : 'erroneous'})}{Pr(\text{judgement : 'erroneous'})} = \\ &= \frac{\text{sensitivity} \cdot Pr(\text{access erroneous})}{Pr(\text{judgement : 'erroneous'})} = \frac{yp}{q} \end{aligned}$$

The latter does not converge, since the limit would depend on whether p or q converges more rapidly to zero. This proves the first claim of the theorem. (Note that the problem of a low PPV arises if and only if $p \ll q$, which condition also implies that the validator gives many false positives.)

To prove the second claim, the last expression is transformed so that it contains also x but not q:

$$PPV = \frac{yp}{yp + (1-x)(1-p)} = \frac{1}{1 + \frac{(1-x)(1-p)}{yp}}$$

Obviously, p depends neither on x nor on y. So the partial derivatives are:

$$\begin{aligned} \frac{\partial PPV}{\partial x} &= \frac{1-p}{yp} \left(1 + \frac{(1-x)(1-p)}{yp}\right)^{-2} \to \frac{1-p}{p} \\ \frac{\partial PPV}{\partial y} &= \frac{(1-x)(1-p)}{y^2 p} \left(1 + \frac{(1-x)(1-p)}{yp}\right)^{-2} \to 0 \end{aligned}$$

which proves the theorem. □

Finally, in order to further justify the result that PPV depends heavily on the specifity but hardly on the sensitivity (at least in the region in which they are both high), figure 3 shows PPV as their function.

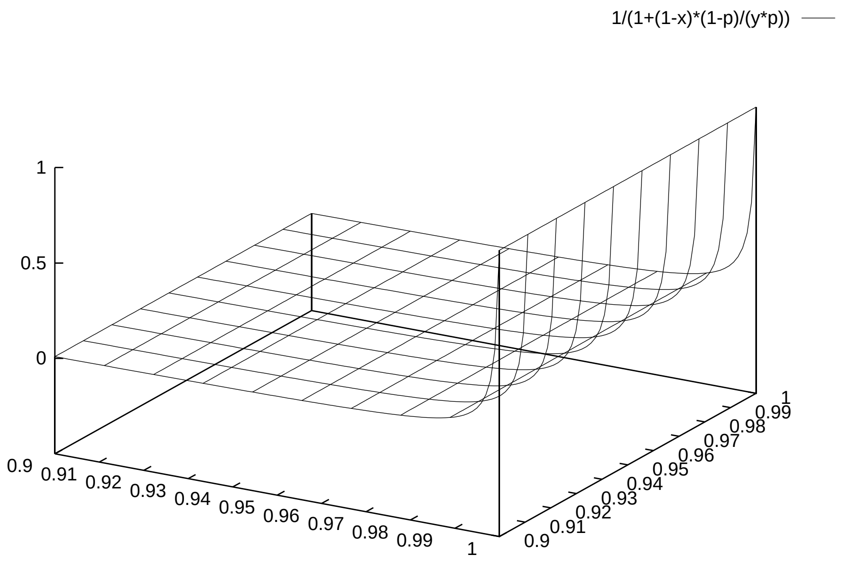

Figure 3: The positive predictive value as a function of the specifity (x) and the sensitivity (y). In this example, $p = 10^{-3}$ was used.

4.2 Non-functional quality measures

Beside the requirement that the validator should give correct judgements, there are other, non-functional, requirements that must also be met. To summarize: the 'price' that must be paid for the automatic validation should be kept as low as possible. This 'price' is in many cases not directly quantifiable, so its determination can be even more problematic than that of SC. The following paragraphs give a brief overview on the most important non-functional quality measures of a validator.

Efficiency. The validator should work fast, and it should burden the resources of the executing system as little as possible. Particularly, the validation should not make the work of the mediator system significantly slower or more resource-intensive. Consequently, only relatively simple validation methods can be used on-line. If more complicated methods are needed, they must be run off-line (*e.g.* when the system is unused).

Programming expenditure. The validator causes extra programming expenditure, which depends of course heavily on the complexity of the validator. Naturally, not only the construction of the validator has to be taken into account, but also its maintenance.

Generality. The validator should be wrapper-independent, as much as possible. That is, the validator should not make much use of the way the wrapper works. Otherwise, every change in the wrapper would make it necessary to change the validator as well.

Reusability. Reusability is in strong connection with generality, but it means more than that. Reusability means in this context that the validator or parts of it can be reused with other wrappers and ISs.

Adaptability. Adaptability is hard to define and it is hard to achieve. It means that the validator can adapt to changes in the output of the IS. Suppose for instance that a wrapper extracts names of persons, and the IS has always given full names. If the IS is changed so that it returns abbreviated first names, a validator operating on the syntax level will probably signal an error. An adaptive validator on the other hand would adapt to the new circumstances.

Total cost. In a commercial environment it is common to just look at the bottom line and take only the overall costs into account. This is made up of the costs for the design, implementation and maintenance of the validator as well as the costs of any modifications of the mediator system that were necessary to host the new validation functionality (see also section 6).

5 Validation methods

This section presents various validation mechanisms, classified according to the principal concept of their way of working. At the end of the section a comparison of the presented methods is given. The practical applicability of the suggested validators will be covered in section 7.

5.1 Validation while the wrapper is extracting

The first group of validation schemes validate the way the wrapper works, and not only its output. These can be regarded as white-box test methods. The most important advantage of these methods is that at this point all information is available, whereas if only the output of the wrapper is investigated, the state information of the wrapper (S_W in definition 4) is lost. For instance, as already mentioned, if the wrapper output is ε, this might be the result of an error but it can also be a valid reply of the IS; if the way the wrapper has worked is investigated, it might be possible to differentiate between these two cases.

The most important disadvantage of such methods is their high wrapper-dependence. Thus, if the wrapper is modified for some reason, the validator also has to be updated. It is also possible that a slight change in the wrapper makes the validator completely and unrepairably unusable. This can increase maintenance costs dramatically. If the wrapper and the validator are tightly coupled, this makes the validator – and even its components – hardly reusable.

Because of this high wrapper-dependence it is virtually impossible to describe general methods of this kind. Instead of that, we give some examples that may be used to validate MetaSearch:

- Detecting purely technical errors, *e.g.* that the page could not be fetched;
- Detecting that the RegExp engine used in the wrapper takes an abnormally long time to extract from the page, indicating that something has changed;

- Detecting that the finite automaton of the RegExp engine behaves unusually, *e.g.* it visits states in an unusual order.

It is possible to use some of these wrapper-dependent features in the validator and maintain a relatively high degree of wrapper-independence at the same time. This requires a well-defined interface between wrapper and validator that prescribes for the wrapper how it should make these data available. In this case, the validator can use these pieces of information as if they were part of the wrapper output. This way, the wrapper can still be regarded as a black box. However, this approach puts restraints on wrapper construction and modification because the wrapper has to comply with the prespecified interface.

5.2 Validation of the output of the wrapper

The methods presented in this subsection make no use of information internal to the wrapper, only its output. This can be thought of as black-box testing. The advantages and disadvantages are exactly the opposite as before: the complete independence of the internal state of the wrapper is advantageous; however, the lack of this information may cause problems. As explained in the Introduction, most existing results use this paradigm for wrapper validation.

The principal idea behind these methods is that the wrapper output takes its values normally from a real subset of Σ^* (denoted by L), *i.e.* certain strings from Σ^* are not plausible. In some cases it might be a better idea to define L to be a fuzzy set [14] because the border between plausible and not plausible wrapper outputs may not be clear-cut. There might be wrapper outputs that are very unlikely but theoretically possible.

So the task of the validator can be defined as follows: given a wrapper output r, is $r \in L$? In order to answer such questions, the validator must contain the specification of the set L. If L is small, the validator can have a list of all plausible wrapper outputs. If, on the other hand, L is large, possibly infinite, other methods are needed. The theory of formal languages [1] has invented several methods to represent large, possibly infinite sets finitely and compactly, supposed that the given set is structured enough. For example, a regular grammar, or equivalently, a finite automaton can be used in some cases to encode the set L.

However, the classical results of the theory of formal languages are in many cases insufficient. The most important problems include:

- Uncertainty and fuzzy borders have to be taken into account;
- It is often very tough to create a grammar or an automaton for a given language;
- Some languages cannot even be represented by a finite grammar or automaton;
- Even if a grammar or automaton can be constructed, it is often too resource-intensive.

In such problematic cases other, *ad hoc* methods can be used. The validation task can be interpreted as a classification task. There are basically two methods to solve it:

- A procedure is implemented that specifies somehow the elements of L. To do so, for example binary decision diagrams (BDDs), fuzzy calculus, and various approximations can be used. In many cases a set L' can be found that is almost the same as L but can be characterized more easily. For instance, it may be possible that L can be made context-free with a small modification although it is context-sensitive. This way, the costs of the validation can be reduced substantially, and if the difference between L and L' is not too big, the correctness of the validation decreases only insignificantly, thus yielding a better trade-off between accuracy and costs.
- The classification task can also be solved automatically, using a machine learning approach. Ideally, the machine learning algorithm is trained with positive and negative examples, *i.e.* with elements from L and from $\Sigma^* \setminus L$ (and each example is labelled accordingly), so that the algorithm can learn the border between the two sets precisely. However, there are also machine learning algorithms that learn from positive examples alone (see e.g. [20]). Such algorithms are more suitable for application in the IAVP domain.

Both classes of methods can detect mainly syntactic errors. Both make use of some regularity of the set L; if no such regularity exists, there seems to be no better way than simply listing all elements of L.

Learning the definition of the set L automatically certainly has the advantage of smaller programming expenditure. This is most notable in the maintenance phase: in the case of a change in the set L, the automatically created validator only has to be retrained, which may also happen automatically [15]. Automatically created validators also tend to be more flexible and more general. On the other hand, even if the utilized learning algorithm is proven to converge in the limit, if often yields imperfect solutions after a finite number of steps. Thus, manually constructed validators are usually more reliable. (Notice that these are the usual pros and cons in connection with machine learning so they apply also *e.g.* for automatically vs. manually created wrappers.)

Classification can often be made easier using a carefully chosen transformation. Let $T : \Sigma^* \to X$ be a transformation; the image of L is $T(L)$. T is to be considered useful if $T(L)$ is more regular and thus easier to specify than L (*i.e.* classification in X is easier than in Σ^*). A very important special case of such a transformation is the usage of *features*. This usually involves a heavy reduction in dimension, because usually only a handful of features is used: if the number of features is k, then X is a k-dimensional space. If it is further assumed that the feature values are real numbers, then $X \cong \mathbb{R}^k$. If $T(L)$ is very small, its elements can even be listed in the validator. However, the goal of the reduction in dimension is usually not that the resulting set should be as small as possible, but rather to increase its regularity. What puts a limit to the application of this scheme is the loss of information caused by the reduction of dimension, which may result in degraded specifity and sensitivity. Therefore, an optimal trade-off between these two effects is needed.

A typical example of the usage of features is the algorithm RAPTURE proposed by Kushmerick [16, 18]. It uses statistical features of the extracted tuples such as average word length, number of words, and number of special characters. The implicit assumption behind the algorithm is that $T(L)$ is a convex subset of $\mathbb{R}^k$. An important advantage is that these features are problem-independent and thus the gained validator can be deployed in various domains.

At the end of this subsection, we summarize the sources of inaccuracy in the algorithms that operate on the wrapper output:

- It is not necessarily clear from the wrapper output whether or not an error occurred during the access to the external IS;
- In order to make classification easier, the aimed set L is often approximated with a similar set L';
- Possible bugs;
- The border between L and $\Sigma^* \setminus L$ as specified by machine learning may not be perfect;
- A reduction in dimension results in loss of information.

Nevertheless, these methods can relatively efficiently filter out many errors, so they often represent the best alternative (see section 5.6 for a detailed comparison of validation methods).

5.3 Validation using additional ISs

As already mentioned in section 3, the detection of semantic errors requires additional, more or less domain-specific knowledge. In a mediator system the most natural way to integrate additional knowledge is the integration of new ISs. These ISs can then be used by the validator to determine if the output of the accessed IS makes sense in the given context. Hence, redundancy is achieved, which can generally be used to improve system reliability [24]. Several types of additional ISs may be used:

- ISs that are already integrated into the mediator system. This kind of reuse helps keeping validation costs low;
- ISs which are integrated into the system specifically for this purpose. The corresponding additional wrappers (and theoretically the corresponding validators as well) have to be implemented. This can increase construction and maintenance costs significantly;
- the user themselves. They can either be asked directly if the output makes sense, or their opinion can be deduced from their behaviour. For instance in the case of MetaSearch, if the user clicks on the offered URL, this is a sign that they found it appropriate. If the human-computer interface allows, also their facial expression can be used to determine their opinion [26]. Their verdict can either simply override the judgement of the validator, or, in a more complex learning scheme (*e.g.* reinforcement learning [13]) it may be used as a reward or penalty for the validator.

5.4 Validation using test queries

In testing conventional software systems, regression tests play a vital role [16]. They involve comparison of the output of the system with a known correct output for various inputs. Unfortunately, this scheme cannot be used directly in the case of the IAVP, because it is by no means guaranteed that the IS answers the same query always with the same reply. That is, if the reply of the IS is not the expected one – which is an error in regression testing –, the access was not necessarily erroneous. Hence this method can only be applied to static ISs. Kushmerick reports in [18] that only 19% of the sites he investigated possessed this property (*e.g.* the Bible and the constitution of the United States). For the remaining 81% this method is not applicable directly. However, several similar methods may well be used:

- As already mentioned, it is an important task to determine whether an empty output means that the IS has correctly returned ε as reply, or the wrapper has ceased to work properly because of a change of the IS. In this case a test query can be used for which the IS will surely return something other than ε. In the case of MetaSearch for instance, the query 'Internet' can be used, because every search engine will return lots of pages for it.
- If no such query is previously known, it can be collected during system operation. In the case of most ISs it is unlikely that a query that has just produced many results will suddenly produce an empty reply.
- Even if the IS is greatly autonomous, it can sometimes be achieved that it return a well-defined reply for a certain query. For instance, if a Web page is created with a nonsense word as title, and the page is submitted to search engines, they will produce that page if the particular nonsense word is queried.
- Traditional regression test can be applied if the error criterion is somewhat relaxed: instead of perfect match of the old and the new output, only similarity should be required. The basic assumption behind this idea is that small changes are frequent and should be tolerated, but the probability of a vast change is very low, definitely lower than that of an error. For this method to work well, a careful definition of the similarity of two replies is necessary.

As can be seen, methods in this category can only be used efficiently in conjunction with other validation methods. However, they are very useful in some cases.

5.5 Validation of the mediator system

All methods described above (validation while the wrapper is working, validation of the output of the wrapper, using additional ISs, test queries) can also be used at a higher level, namely to validate the whole mediator system instead of just one wrapper. This is in many cases more practical. If q ISs are integrated into the mediator system, this reduces the number of necessary validators from q to 1. Thus, validation is cheaper, more efficient and also more robust because the validator does not have to be modified after every change in the wrappers or the ISs. In many cases, it may not even be feasible to validate the replies of each IS because the necessary additional knowledge is not available.

As an example, consider a mediator system that can tell how much certain kinds of pizza cost in Italy. For this, two ISs are integrated: the first returns the price of the pizza in Italian Lira (ITL), and the second is used to convert it to, say, USD. If no additional knowledge about the prices in ITL or the exchange rates is available, the only possibility is to validate the overall result. For instance, if the result is that a pizza costs 2 cents, this probably implies that an error has occurred (*e.g.* in the exchange rate table the USD value was given for 100 ITL, and now it is given for 1 ITL, but the wrapper or the mediator still divides it by 100).

The main disadvantage of such methods is that not necessarily all errors are that clear from the output of the mediator system. Thus, many errors can remain undetected.

5.6 Comparison of the suggested validation methods

All of the presented algorithms can be used to detect errors during the access to external ISs in a mediator system. How well the particular methods perform, depends on the specific application and, even more importantly, on the used ISs and wrappers. Therefore, a general comparison is hardly possible. For best performance, the presented methods should be used together. On the other hand, this can increase validation costs substantially.

In most applications where communication is textual (*e.g.* in the WWW) the best choice is probably a syntactic check of the wrapper output, because this is relatively simple, it is not dependent on the implementation details of the wrappers, and – as a result of the redundancy introduced by textual communication – many errors can be detected with such methods. Unfortunately, these methods can be quite inaccurate in some cases. This holds for both validation of access to a single IS and also validation of the whole mediator system.

If these validation methods are not enough, more intelligent methods must be used. In this case, the integration of additional ISs should be considered, because this would make it possible to detect semantic errors. However, this will also increase the costs of validation and the overhead generated by validation significantly.

The other methods (test queries and validation during the operation of the wrapper) are rather special, but there are many cases in which they can be used successfully. Test queries should only be used together with other methods because they can only detect a fraction of all possible errors. Validation during the operation of the wrapper may be very effective; however, it makes maintenance quite hard, so it should only be used if absolutely necessary. If possible, an interface should be defined for the wrapper to communicate its inner state.

At the end of this section, table 2 summarizes the advantages and disadvantages of the suggested methods.

6 Integrating validation into the mediator architecture

Several validation mechanisms have been proposed in the last section. However, it is yet to be cleared how this functionality can be best embedded into the mediator architecture. Figure 1 reveals several places suitable for the integration of the validator.

Validation scheme	Advantage	Disadvantage
Validation during the operation of the wrapper	All necessary information is available	Very wrapper-dependent
Validation of the output of the wrapper	Quite general; many errors can be detected with simple methods	Not all errors are clear from the wrapper output; specification of plausible outputs may be hard
Using additional information sources	Also semantic symptoms can be used in the validation process	Validation may become quite costly
Test queries	Established method; in some cases practically the only solution	In many cases inappropriate
Validation of the whole mediator system	Small expense	Some errors can remain undetected

Table 2: Comparison of the presented validation schemes

This section will discuss the most important possibilities. Also note that the choice of the validator and the way it should be integrated into the mediator architecture are not completely orthogonal.

6.1 Validator as part of the wrapper

Since the validator has to deal with the access to the external ISs which is done in the wrapper, it is logical to include the validator also in the wrapper. Thus, wrapper and validator together can be regarded as a more reliable wrapper. Moreover, if the validator uses internal information of the wrapper, they must be tightly coupled. In this setting, if the validator judges an access to be erroneous, it can do one of the following:

- It can inform the user that something went wrong. This should only be done this way if the mediator and the wrappers (and the validators) are built together monolithically, because usually the mediator should be the only component of the architecture to interact with the user.
- It can inform the mediator that something went wrong. In turn, the mediator can decide if additional tests are needed, if the user should be notified, or if reparation should be attempted. The wrapper cannot use the standard communication schemes of the system for such messages, but it may return an error code (see section 6.5).
- The wrapper is automatically repaired. Of course this might not succeed so that either the user or the mediator may still have to be notified.
- The validator informs the programmer in charge of maintaining the wrapper so that they repair it.

- The validator makes the wrapper repeat the access.
- The validator makes additional tests to have a better and more confident understanding of the error.

If the validator needs additional knowledge for the validation, this can either be situated inside the validator, or using an external IS. In the latter case, the IS must be accessed through the mediator (and through another wrapper, which should theoretically also be validated), using the standard communication schemes of the system.

6.2 Validator as part of the mediator

The validation functionality can also be integrated into the mediator itself. This seems to be useful if the whole mediator system is validated. Otherwise it may be better from a software engineering point of view to decouple the validator from the mediator.

If the validator is part of the mediator, the communication between the validator and the user or the programmer in charge is best arranged through the mediator. Also, if an external IS is needed, it can be accessed in a straight-forward way through the mediator. On the other hand, wrapper-internal information can only be used with significant overhead.

6.3 Validator as a separate component

From a software engineering point of view, the best solution is generally the separation of the validator from the other components of the mediator architecture. Nevertheless, this realization requires the construction of additional communication channels, through which the validator can obtain the information it needs and it can communicate its judgement. This increases the implementation costs of such a solution; however, it results in a cleaner and more modular structure which can decrease maintenance costs and improve reusability.

6.4 Validator as an IS

An important special case of the method just described is when the validator is not only treated as a separate component but also as an IS. The basic idea behind this approach is that the validator also implements a function similar to the one associated with ISs: it can be queried from the validator whether the access (the data of which are given as input to the validator) was correct or not. (See the definitions in section 2.)

The main advantage of this approach is that this way the integration of the validation functionality does not increase the complexity of the mediator architecture. Validators are handled the same way as other ISs except that their input is not derived from the query posed by the user, but the output of the wrappers. Also, its output is interpreted differently. Note that the validator is not autonomous like other ISs, and thus no wrapper or validator is needed to process the output of the validator.

A disadvantage is that the validator cannot make use of other ISs directly because the mediator architecture does not normally support interaction between individual ISs.

6.5 Error codes

As already mentioned, the validator may return its output as an error code. Whether it is indeed numerically encoded, depends on the particular application. The error code may contain the following information:

- Whether or not the access was erroneous
- The confidence level of the judgement
- Possible reason(s) of the error
- Recommended counter-measure
- A set of other tests that could be conducted to get more information on the error or to increase the level of confidence
- Cost estimates (time, money, resources, *etc*.) for these tests

6.6 Diagnosis function

In the preceding paragraphs, generally on-line validation was considered, *i.e.* it was assumed that the validation takes place directly after (or even during) the access. However, virtually the same methods can be applied in an off-line scenario as well, *i.e.* if validation takes place in phases in which the system is not working (*e.g.* during the night). The validation algorithm can be the same, only the integration into the mediator system is different: data is collected – in the wrapper or in the mediator, or at both places – in the daytime, to be validated during the night. (Of course there might also be a predefined set of test data, not only the ones collected in the daytime.) This kind of validation scheme is particularly useful if the validation takes too long to be tolerated during normal operation of the system.

If such a validator is implemented in the wrapper, it can be regarded as a diagnosis function, with which it can be queried asynchronously whether the wrapper still functions properly – based on the collected set of data. Asynchronicity means that this function can be invoked at any time later, not necessarily directly after a certain access. Similarly, if the validator is realized in the mediator, it can be regarded as a diagnosis function, with which it can be queried asynchronously whether the whole mediator system still functions properly. It is also a possible strategy to invoke the diagnosis functions periodically, if the system still performs well.

6.7 Hierarchical, multi-level validation

As already mentioned, more than one of the validation methods suggested in section 5 can be used in a given application. Moreover, the usage of more than one validation method can sometimes be very useful. It is also possible that the individual validation methods are not implemented in the same component, but rather form different validation levels.

For instance it may be useful to include a validator in each wrapper that performs a cheap but superficial check after each access, and a more accurate, but also more resource- and time-intensive validator to provide a diagnosis function. Moreover, an additional validator that validates the whole mediator system can be integrated as an IS *etc*.

7 Case study

In this section we demonstrate the practical applicability of the above methods on the example of MetaSearch. Recall that MetaSearch is important for two reasons: first, it is a relatively simple application implemented using KOMET that nevertheless exhibits most of the problems that arise during the access to external ISs; and second, it can be regarded as a representative of the increasingly important class of Web-based information systems.

Several validation methods have been implemented and several experiments have been conducted in order to:

- demonstrate the applicability of the suggested methods
- compare the suggested methods
- compare the results with those in the literature

On the other hand, the implementation did not aim at creating a perfect validator that could generally be integrated into KOMET. This might be the goal of a future project. For our goals it was easier to first construct a simplified version of MetaSearch, and then conduct the experiments on this stripped-down version, which has the following characteristics:

- Various Internet search engines have been integrated into the software using specialized wrappers
- The user (which might be another program) can pose a query
- The user can choose a search engine
- The query of the user is sent to the specified search engine
- The returned page is processed in the wrapper, and the titles, URLs and extracts are extracted

That is, the mediator-specific details were omitted, and only the features vital for communication and extraction have been adopted. Additionally, the output is validated.

Originally, the Java programming language was to be used for the implementation, complemented with the `gnu.regexp`[5] package to handle regular expressions. The first experiments were conducted in Java, but eventually the Perl programming language has been chosen for the following reasons:

[5] `http://www.cacas.org/~wes/java/`

- it has excellent support for regular expressions, which can be used both for extraction of data from the HTML page and for validating the extracted data;
- fetching a Web page is very easy using the LWP[6] module – also through proxy and firewall;
- it is a perfect tool for rapid prototyping, because also complicated ideas can be tried instantly.

Moreover, some smaller utilities have been implemented as Unix shell scripts, and a simple graphical user interface was constructed using the graphical library Tk.

The next two subsections document implementation details, and test experience, respectively.

7.1 Implementation details

The implementation took place in three steps:

1. An interactive, but simplified version of MetaSearch was created.
2. Several validation mechanisms were implemented.
3. A batch program was written for the test of the implemented validation methods on a large set of data.

In order make experimenting easier, the interactive version of the program provided the following functionality:

- Fetch & Parse: the specified query is sent to the chosen search engine, the returned page is fetched and the data are extracted.
- Fetch & Save: same as above, but the returned page is not extracted, only saved to a given file.
- Load & Parse: a previously saved page can be loaded and processed as if it were currently being returned by the IS as a reply to a given query.

Since the saved pages can be altered before loading using a text editor, this enables many experiments to detect how the wrappers and validators react on specific changes. This way, several interesting experiments have been made, which are described in section 7.2.

The source code of this program was divided into two modules: `main` and `Wrapper`. In module `main` the wrappers and the graphical user interface are created. Module `Wrapper` contains the definition of the class Wrapper. Instances of this class represent particular wrappers, which are specialized by specifying the base URL of the corresponding search engine as well as a regular expression for the extraction of data. Also, this class defines the primitive methods `load_html` and `parse`, which are called from `main`.

In the next step, the following validation methods were implemented:

[6] `http://www.cpan.org/modules/by-module/LWP/`

- A syntactic validator for the URLs. It would have been possible to construct a parser for the language of URLs[7], but it seemed easier to just implement a regular expression for it. It is not perfect, but it performs very well in practice, and it is much more compact. Also, this makes it possible to fine-tune the specifity and sensitivity of the validation by making the regular expression more accurate or by relaxing it. In our tests, the following expression was used: (in Perl syntax)
  ```
  ^((http\://)|(ftp\://))?[-\+\w]+(\.[-\+\w]+)+(\:\d+)?\
  (/\~[-\+\.%\w]+)?(/[-\+\.\,\:%\w]+)*/?(\?([-\+\.%\w]+\
  =[-\+\.%\w/]+&)*[-\+\.%\w]+=[-\+\.%\w/]+)?$
  ```

- A syntactic validator for the titles. For this, a very simple but practically well performing method was used: the title can be anything, but it is always relatively short. Although it is not declared explicitly in the official HTML specification[8] that the length of the title would be limited, but since the title is usually displayed in the title bar of the browser and long titles cannot be displayed, long titles are very rare. Tim Berners-Lee[9] recommends [2] that the title should be at most 64 characters long. The validator checks if the length exceeds 100 characters.
  On the other hand, the specification of the HTML language declares that the title cannot contain any formattings (*i.e.* HTML tags), so the validator could theoretically also check this. However, the titles extracted from the output page of the search engine might still contain tags. Some search engines use for instance italic or bold characters do mark the words of the query if found in the title or in the excerpt. Therefore, this characteristics is not checked by the validator.

- A similarly simple, syntactic validator for the excerpt: the length of the excerpt must not exceed 300 characters. Of course, this value could not be obtained from the HTML specification, since syntax and semantics of the extracts are defined by the search engines. However, these definitions are not public, so this value had to be determined empirically. It is also required that the excerpt should contain at least 1 character. This is almost always the case, save for cases in which the crawler of the search engine has not processed the particular page and only knows about its existence from the links that point to the page [3]. In such a case the search engine cannot provide an excerpt of the page; however, such cases are rather rare.

- A more time-consuming semantic validator for the wrapper output. For this, the Internet itself was used as additional IS. Naturally, in order to create a semantic validator, it has to be considered what the actual semantic of the extracted tuples is. In the case of a search engine the semantic can be put something like this: 'The URLs point to pages that probably exist, at least they existed at some time. The pages have probably the given titles, at least they have had them at some time. The extracts have extracted from these pages. The pages contain information that is probably related to the query, they are even likely to contain some words of the query.'
  A semantic validator has to check exactly these characteristics. For this, the Internet can indeed be used: it can be checked if the pages really exist, if their titles are as expected, if they contain the words of the query and of the excerpt, and so

[7] `http://www.cs.ucl.ac.uk/staff/jon/book/node166.html`

[8] `http://www.w3.org/TR/html401`

[9] Father of the World Wide Web and head of the World Wide Web Consortium; for more details see `http://www.w3.org/People/Berners-Lee`

on. On the other hand, as it can also be seen from the description of the semantics above, a big amount of uncertainty is involved in doing this. The pages may have been removed, or their titles and their content may have changed. In order to minimize the caused uncertainty, the validator implements only a simplified version of this test: it is only checked whether the URLs are real in the sense that they point to existing pages. The other parts of the test were omitted because they did not improve its performance.

Accordingly, the class Validator implements the functions `is_URL_OK`, `is_Title_OK`, `is_Excerpt_OK` and `is_Tuple_OK`. Their implementation is straight-forward except for the last one. This is complicated by two factors:

- If the page cannot be found, it may take a long time until `LWP::Simple::get` will give up. In order not to make the validation process too time-consuming, fetching should be canceled after some time t_c. Due to a bug in LWP[10], this could only be done reliably in a separate Unix shell script (`timeout_wget.sh`), using the Unix utility `wget`.
- If n tuples are extracted, n pages have to be checked. Of course, this should be done parallel. Therefore, n copies of `timeout_wget.sh` have to be started. Each one starts a `wget` process in the background and waits for a time period of t_c. After that, if the `wget` process is still working, it is canceled using `kill`.

Each `wget` process tries to save the page it has fetched to a different temporary file. The main program waits for a period of $t_w > t_c$, and afterwards `is_Tuple_OK` checks how many of the pages could be stored in the temporary files.

The third step of the implementation consisted of creating a batch program that tested the realized validation methods on a sufficiently large set of real-world test examples. Since wrapper errors are relatively rare, it is by no means easy to collect a set of test cases that contains enough erroneous accesses. Fortunately, Dr. Kushmerick of Department of Computer Science, University College Dublin has made the data he had collected in the period May – October 1998 [16, 18] available. This set of data contains the reply pages of 27 actual Internet sites on a couple of specific queries, collected for 6 months.

For the evaluation of the implemented validation methods the replies of AltaVista, Lycos and MetaCrawler on the query 'happy' have been selected. This the test suit contains about 200 pages. In the given period of time there were two changes of AltaVista, one in the case of Lycos and two in the case of MetaCrawler. Accordingly, three wrappers have been constructed for AltaVista, two for Lycos, and three for MetaCrawler. Each page was processed with the corresponding correct wrapper as well as with the other wrapper(s) of the particular search engine, thus testing how the validators perform on correct and erroneous wrapper outputs. This amounted to about 440 test cases.

The test program was rather straight-forward. Altogether eight wrappers had to be constructed, and the test pages were also sorted into eight different directories. An array contained the (wrapper, directory) pairs to be tested. For each such pair, all files of the specified directory were processed with the particular wrapper. Validation was invoked automatically by the wrapper, after the information had been extracted from the page.

[10] See *e.g.* `http://www.ics.uci.edu/pub/websoft/libwww-perl/archive/1999h2/0260.html` or `http://archive.develooper.com/libwww@perl.org/msg01318.html`

7.2 Test experience

Interesting experience has been made with both the interactive simplified version of MetaSearch and the batch test program.

During the use of the interactive program, several changes were encountered in the case of AltaVista:

- At the beginning of the observation period (February 2001), the following regular expression was used to extract information from the replies:
  `<dl><dt><b>\d+\.  </b><a href=”(.+?)”><b>(.+?)</b></a><dd>`
- In the early days of March 2001, the output of AltaVista was changed in two ways. The first `<b>` tag took the form `<b class=txt2>`. Also, the order of the `<a...>` and `<b>` tags, and similarly, the order of the `</a>` and `</b>` tags around the title has been reversed.
- Later on in March there were two more changes. The `<a...>` tag was complemented with an `onMouseOver` script, and the URLs obtained a `/r?r` prefix. That is: since then the user is directed through a local script of AltaVista to the actual target page. Thus, AltaVista can gain valuable information about the users' actions. Moreover, the `onMouseOver` script is used to display the original (non-local) URL in the status line, so that the user does not notice the change.
- At the beginning of May, several changes were observed again. The script that took the queries used to be `http://www.altavista.com/cgi_bin/query`; since that time it is `http://www.altavista.com/sites/search/web`. The script `r`, through which the user is directed if they click on a link, was provided with additional arguments as well: `ck_sm` and `ref`. Moreover, the tuples are since then itemized and not numbered.
- At the end of the observation interval (end of May 2001), the following regular expression was to be used:
  ```
  <dl><dt>&#149;<b><a href=”/r?ck_sm=.*?&ref=.*?&r=(.+?)”
  \
  onMouseOver=.*?>(.+?)</a></b>
  ```

The results of the batch test are described in full detail in [21]. In these tests, the following five figures were stored for each test case: the number of extracted tuples, and the number of likely errors found by the four implemented validation methods.

In the case of AltaVista, if the correct wrapper is used, the number of extracted tuples is always 10. In most cases all pages are found to be syntactically correct: the syntactical validators find 0 error. There are only a few cases in which one of the excerpts is empty, which is considered to be an error. On the other hand, the number of non-existent URLs is very high, varying from 10% to 70%, with an average of 40%. Note though that the used data are three years old and it is a common phenomenon that almost half of the pages that existed three years ago cannot be found now (for details, see [19]). It can be expected that the situation is much better with current pages; for justification, see below.

When processing the pages of the first group with the second wrapper, something unexpected happens: the wrapper extracts always exactly one tuple, which consists of a both syntactically and semantically correct URL, a correct title and an erroneous excerpt. The reason is that the second wrapper expects the excerpts to be embedded in the page slightly differently, and it does not find the end of the first excerpt, and thus it interprets the whole page as the excerpt. Therefore it can extract only one tuple, which has a correct URL and title, but its excerpt is too long.

In the other cases where pages from AltaVista are processed with an incorrect wrapper, nothing can be extracted. This is so because the layout had been changed in such a way that the regular expression of the old wrapper can find no match.

The results with Lycos are virtually the same: when the correct wrapper is used, always 10 tuples are extracted, which are all judged correct by the syntactic validation methods, but about half of the URLs do not exist anymore. If an incorrect wrapper is used, no tuple can be extracted.

The results with MetaCrawler are much more varied, because the output of MetaCrawler is generally much less structured and regular than that of AltaVista or Lycos. Accordingly, even the tuples extracted by the correct wrappers are sometimes judged erroneous by the validators. The number of syntactically erroneous URLs varies from 5% to 15%, that of titles from 5% to 20%, and of excerpts from 0% to 40%. The reason for the high number of too long excerpts is that, if a page could be found by more than one search engines, MetaCrawler tends to merge the excerpts returned by those search engines. Also, the number of extracted tuples varies from 14 to 21. On the other hand, the number of URLs not accessible anymore is lower than in the case of AltaVista and Lycos: it varies form 15% to 40%, with an average of about 30%. The reason of this phenomenon may be that MetaCrawler gives preference to pages that could be found by more than one search engines and thus is likely to be more stable.

If a wrong wrapper is used, in most cases either no tuple can be extracted at all, or there is a purely accidental match, which results in a single tuple that is incorrect in each investigated aspect. But if the pages in the second group are processed with the first or third wrapper, 20 tuples are extracted, that are mostly judged correct concerning title and excerpt but incorrect concerning URL. This is caused by the fact that only the embedding rule for the URL was changed, and in such a way that extraction remained possible, titles and excerpts were correctly extracted, only the URLs were erroneous.

The most important conclusion to be drawn from the above results is that the implemented, intentionally very simple validation methods can be combined to a virtually perfect validator. Namely, there is a big difference in the behaviour of the utilized methods in the case of correct and incorrect wrapper outputs. In the erroneous case, the wrapper output was either empty, or at least one of the applied methods (but often all of them) judged all tuples to be erroneous. In the case of a correct wrapper output, the number of tuples judged to be syntactically incorrect was at most 40%, and of those judged to be semantically incorrect at most 70%. Moreover, this last figure is actually much lower normally, when current data are considered (see below). Thus, there is indeed a wide gap in the behaviour of the validation methods in the correct and incorrect cases.

It is also interesting to investigate how much the individual validation methods contribute to the overall excellent performance. It can be seen from the results that two of the four methods – namely the syntactic validation methods for the URL and for the excerpt – would have been sufficient to distinguish between correct and incorrect wrapper outputs. But of course other errors are also possible for which the other tests are necessary. Fortunately, the other methods have not degraded the performance either, and it seems that with a higher number of validation methods it is more probable that future errors can also be detected. It is also important to have at least one validation method for each element of the tuples so that errors that concern only that particular element can also be detected.

In the theory of reliable systems (see *e.g.* [24]) it is proven that, with redundancy and proper combination strategies, a reliable system can be built from unreliable components (*e.g.* Triple Modular Redundancy). The method presented here is an example of this: many simple and unreliable validators can be combined to form a very reliable validator. (Note that the word 'redundancy' has already been used twice in section 5, referring to redundant ISs and a redundant output of an IS. This time, the validators themselves are redundant. All three kinds of redundancy can help make the mediator system as reliable as possible.)

All of the implemented methods are very simple, but the programming expenditure and overhead associated with them is different. The validation methods for the title and the excerpt were very simple to implement and hardly pose any overhead on the system. The syntactic validation method for the URLs took a little bit more effort to implement, and it also generates some overhead, which is still negligible compared to the time that is needed to access the external ISs. In the case of the semantic validator the overhead is significant. Its implementation would have been straight-forward, there were only two complications: a time-out mechanism had to be implemented, and the access to different pages was done parallel.

It has to be noted that the principles behind MetaSearch and MetaCrawler are very similar, so validation of the output of MetaCrawler can also be regarded as the prototype of validating a whole mediator system. Hence, it is clear from the tests that the suggested methods are also appropriate for the validation of the whole mediator system.

Ultimately, we have conducted another set of experiments with a smaller set of current data. The goal was twofold:

- to check if the results based on the three-year-old data set can be transfered to the current situation;
- and to identify trends that may have an impact on the future validation possibilities.

It has to be noted that the set of changes of the output of search engines in 1998 and in 2001 (see above) was very similar. So, it seems that these changes are caused by typical human behaviour, which is quite constant, rather than by some technology because that varies rapidly. Consequently, similar changes are likely to occur in the future as well. This is bad on the one hand, because it means that wrappers and validators will be needed in the next years as well. On the other hand, this is advantageous because it makes possible to construct validators that can be used robustly for many years.

Thus it seems that the implemented syntactic validation methods could be used for at least three years. But, as already mentioned, in the experiments with data from 1998, many URLs could no be found. So we checked how many of the URLs currently

Search engine	Number of non-existent URLs
AltaVista	0%
Lycos	0%
MetaCrawler	10%

Table 3: Results of the tests with current data

returned by the three search engines for the query 'happy' are invalid. Table 3 shows the results.

It follows that this validation mechanism can also be used very effectively.

8 Conclusion

In this paper, we have presented the general information access validation problem (IAVP) that arises in mediator systems, when making use of the integrated external, autonomous ISs. A notational framework was presented that allows the investigation of the problem in a setting as general as possible. This generalizes past results from the literature [6, 15, 18, 20]. We have proven that the general IAVP is not solvable algorithmically in its most natural form, so we presented quality measures that make it possible to quantify imperfect, but in most cases properly working validators. We have investigated the connection between the most important quality measures, namely specifity, sensitivity and positive and negative predictive values.

In the other, more practically-focused part of the paper, the typical errors and solutions were collected. It was also investigated how the validation functionality can be included in the mediator architecture. Moreover, to illustrate the practical applicability of the presented algorithms, some validation schemes have been evaluated on MetaSearch, a simple, but real-world example mediator system.

References

[1] I. Bach. *Formális nyelvek*. TypoTeX, Budapest, 2001.

[2] T. Berners-Lee and D. Connolly. RFC 1866: Hypertext Markup Language 2.0. `http://www.ics.uci.edu/pub/ietf/html/rfc1866.txt`, November 1995.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *WWW7/Computer Networks*, 30(1-7):107–117, 1998.

[4] J. Calmet, S. Jekutsch, P. Kullmann, and J. Schü. KOMET – A system for the integration of heterogeneous information sources. In *10th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, 1997.

[5] J. Calmet and P. Kullmann. Meta web search with KOMET. In *International Joint Conference on Artificial Intelligence, Stockholm*, 1999.

[6] W. W. Cohen. Recognizing structure in web pages using similarity queries. In *AAAI-99 (Orlando)*, 1999.

[7] I. Dési. *Népegészségtan*. Semmelweis Kiadó, Budapest, 1995.

[8] V. J. Easton and J. H. McColl. Statistics glossary. `http://www.cas.lancs.ac.uk/glossary_v1.1/main.html`.

[9] Federal Information Processing Standards Publication 161-2. Announcing the Standard for Electronic Data Interchange (EDI). `http://www.itl.nist.gov/fipspubs/fip161-2.htm`, April 1996.

[10] T. Goan, N. Benson, and O. Etzioni. A grammar inference algorithm for the world wide web. In *Proc. of the 1996 AAAI Spring Symposium on Machine Learning in Information Access (MLIA), Stanford, CA*. AAAI Press, 1996.

[11] C. Hsu and M. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Journal of Information Systems*, 23(8):521–538, 1998.

[12] Intelligent Information Integration, Inofficial home page. `http://www.tzi.org/grp/i3`.

[13] L. P. Kaelbling, M. L. Littmann, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[14] G. J. Klir, U. St. Clair, and B. Yuan. *Fuzzy set theory – foundations and applications*. Prentice Hall, 1997.

[15] C. A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: a machine learning approach. *Data Engineering Bulletin*.

[16] N. Kushmerick. Regression testing for wrapper maintenance. In *AAAI-99 (Orlando)*, pages 74–79, 1999.

[17] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence Journal*, 118(1-2):15–68, 2000. (Special issue on Intelligent Internet Systems).

[18] N. Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000. (Special issue on Web Data Management).

[19] S. Lawrence, D. M. Pennock, G. W. Flake, R. Krowetz, F. M. Coetzee, E. Glover, F. A. Nielsen, A. Kruger, and C. L. Giles. Persistence of web references in scientific research. *IEEE Computer*, 34(2):26–31, February 2001.

[20] K. Lerman and S. Minton. Learning the common structure of data. In *AAAI-2000 (Austin)*, 2000.

[21] Z. Á. Mann. Dynamische Validierung von Zugriffen auf externe Informationsquellen in einer Mediatorumgebung. Master's thesis, Universität Karlsruhe, Institut für Algorithmen und Kognitive Systeme, 2001.

[22] I. Muslea, S. Minton, and C. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 2001.

[23] D. J. Power. What is a Decision Support System? *The On-Line Executive Journal for Data-Intensive Decision Support*, 1(3), October 1997.

[24] B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, editors. *Predictably dependable computing systems*. Springer-Verlag, Berlin, 1995.

[25] S. Soderland. Learning extraction rules for semi-structured and free text. *Machine Learning*, 34:233–272, 1999.

[26] C. Stephanidis and M. Sfyrakis. Current trends in man-machine interfaces. In *ECSC-EC-EAEC, Brussels*, 1995.

[27] V. S. Subrahmanian, S. Adalı, A. Brink, R. Emery, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, and C. Ward. HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.

[28] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[29] G. Wiederhold. Value-added mediation in large-scale information systems. In *Proceedings of the IFIP-DS6 Conference*, 1995.