ESTRELLA

IST-2004-027655

European project for Standardized Transparent Representations in order to Extend LegaL Accessibility
Specific Targeted Research or Innovation Project

Specific Targeted Research Project
Information Society Technologies

Deliverable No. D1.3. – Translators to and from the LKIF for each of the knowledge formats of the participating vendors

Version:	Revised version
Due date of Deliverable:	month 24
Actual submission date:	January 31t, 2008
Start date of Project:	1 January 2006
Duration:	30 months
Project Coordinator:	Universiteit van Amsterdam (NL)
Lead contractor deliverable:	C06, RuleBurst (Europe) Limited
Participating contractors:	C06 - RuleBurst (Europe) Limited C05 - Rulewise B.V. C07 - Knowledgetools International GMBH

Project funded by the European Community under the 6th Framework Programme

	Dissemination Level	
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the Consortium (including the Commission Services)	

Partner List

	Abbrev.	Organisation	Country
1	UVA	Universiteit van Amsterdam	NL
2	UNIBO	Alma Mater Studiorum – Universita di Bologna	IT
3	LIVUNI	The University of Liverpool	GB
4	FHG/FOKUS	Fraunhofer Gesellschaft zur Foerderung der angewandten Forschung E.V.	DE
5	Rulewise	Rulewise B.V.	NL
6	RuleBurst	RuleBurst (EUROPE) LIMITED	GB
7	KI	Knowledgetools International GMBH	DE
8	IDL	Interaction Design Limited	GB
9	SOGEI	SOGEI - Societa General d'Informatica S.P.A.	IT
10	CNIPA	Ministro per l'innovazione e le technologie	IT
11	APEH	Hungarian Tax and Financial Control Administration	HU
12	CUOB	Budapesti Corvinius Egyetem – Budapest	HU
13	BMF	Bundesministerium der Finanzen	DE
14	MEF	Ministerio dell'Economia e delle Finanze	IT
15	CPR	Consorzio Pisa Ricerche SCARL	IT

Contents Sheet

1 Executive Summary

The purpose of the deliverable was for the vendors to produce a translator or translators to allow import and export to and from their systems, and one or more of the LKIF syntaxes. Two vendors, KnowledgeTools and RuleBurst, decided to transform to and from Argument Graphs in the Compact XML representation of LKIF. RuleWise chose to transform from their UML/OCL representation to OWL-DL.

Presently, all vendors have in place a translator that supports at least a large portion of their chosen subset of LKIF. KnowledgeTools and RuleBurst have had to remove support for certain features of their format, either due to time constraints, or a lack of support in LKIF, but none of them are major omissions. Despite this, all vendors are happy that the existing LKIF formats, perhaps with minor adjustments, are suitable for expressing their Rules Structure.

Testing has been done against all the Translators using real source material to model from. Some problems were highlighted during testing, but overall the translators were found to maintain the integrity of the original rules to a suitable level. It is currently the case that all rules imported using the translators will require a certain amount of manual confirmation and modification to ensure their accuracy, but this is considered acceptable, especially in the scope of the deliverable.

2 Introduction and Task Interpretation

With differing Formats between Vendors, a requirement exists for each vendor to produce a way of converting from their format, to a syntax of LKIF. The simplest way for this to occur was with the production of translators which went straight from the source vendor format, to the LKIF format.

The purpose of the task is for the involved Vendors to produce a translator which can take a Rule System in their format, and convert it to and from an LKIF syntax of their choice. No strict tasking was enforced and it was left up to the Rule Vendor to decide which LKIF syntax they felt was the most suitable match for their modelling format.

In addition to this, the format of the translators was not specified and as such it was left up to the individual vendors to produce the translator using the most suitable means available to them.

3 Approach

3.1 RuleBurst

The translator currently being developed by RuleBurst supports the Argument Graph representation in the Compact XML Syntax of LKIF, and supports importing/exporting from PIF files produced in RuleBurst 9.1 and above.

Import
The structure of an Argument Graph is almost identical to the representation of basic rules in Word using RuleBurst. By extracting each Argument from the Argument Graph, we can use the statement attribute from each Conclusion/Premise/Exception to reference back to the list of *s* tags and find the matching Statement. These statements are then inserted, in the order Conclusion followed by Premises/Exclusions, into the PIF format, which can be read in by RuleBurst Studio to construct the Word document.

The absence of either a statement attribute or corresponding *s* tag will cause the import to fail, since not enough information exists to reconstruct the rules effectively in RuleBurst.

Currently a debate remains as to the exact form in which to interpret Exclusion statements. At present these, along with premises for which the polarity is negative, are imported as negated statements. This may change before the review so that exclusion statements are no longer collected during the input of base level facts and must be instead be explicitly stated.

Export
Likewise, the exporting of rules from a RuleBurst rule base is a fairly simple process. RuleBurst Studio identifies premises, intermediate premises, and conclusions in the Export to the PIF XML format. A simple XSLT can be applied to this XML file to transform it into a set of Arguments, extracting out the natural language statements into *s* tags. The only difficulty comes when identifying the negated statements – Whilst this in itself is not an issue, it then becomes a requirement to transform the negative statement text into its positive form using the RuleBurst language Parser. Currently, this interface has not been implemented, but the possibility also exists to extract the positive form of sentences from the PIF file instead. Though this is not a simple process, it is currently being investigated and is expected to be implemented before the review.

Things that are not currently exported

- Variables
    - Argument graphs do not support the idea of Variables, and as such, these are not explicitly exported from RuleBurst. Though sentence text will be maintained, it is unlikely to be immediately useful to anyone subsequently importing or using the argument graphs. Despite this, it is possible to argue against said sentences, and as such they’re kept.

- Entities
    - From the perspective of an argument graph, entities are merely the ability to take a subsection of the graph, and multiply it **n** times, arguing against all or a combination of the subsequent arguments, to prove another static argument. Since LKIF only supports static argument graphs, it is not possible to export this functionality to LKIF unless **n** is already known. Since this is not the case, they are not exported.
- Rules written in Excel
    - Since Excel Rules hinge around the idea of variables, there is little point in even attempting to export these to argument graphs. Though it is possible to write rules in Excel that could feasibly be exported to an Argument Graph, the translator required for such a job would be inordinately complicated, compared to the process of writing said rules in Word.

3.2 KnowledgeTools

When developing the translator for LKIF our main aim was to retain the logical consequences of the knowledgetree respectively the LKIF document and at the same time to minimize the differences between both formats.

The following aspects of knowledgeTools are implemented when exporting a tree into the LKIF format:

- the startnode (the “root” of the tree)
- all proceeding nodes that can be reached from the startnode
- the title and
- the logical conjunction of those nodes

The aspects which are not regarded when exporting a knowledgetree into the LKIF format are amongst other things:

- alternative text and additional information
- detached nodes
- versioning of the tree
- different paths of the same node

Of course, knowledgeTools does not consider all of the aspects of a LKIF document when importing:

- the element <rule> and its children
- the attribute “id” from <argument-graph>
- the attribute “src” from <statement>
- the attribute “id” and “scheme” from <argument>

On the other hand, knowledgeTools does implement:

- the element <statement> with its attributes “id” and “summary”

- the element <argument> in <argument-graph>
- the elements <premise>, <assumption> and <exception> in <argument> with their attributes “polarity” and “statement”

An export from knowledgeTools into LKIF generates one statement for each node, and zero or more arguments for each statement. In our initial design, only “and”, “or”, “not-and”, and “not-or” nodes were assumed to be translatable, while “xor” and “not-xor” nodes are not permitted by the translator.

On import from LKIF to knowledgeTools, at least one node is created for each statement, but additional “blank nodes” can be needed.

Due to differences between LKIF and knowledgeTools, it is expected that the following situation can occur: When importing a file a.lkif as a tree A, and when exporting that tree back into a file b.lkif, the files a.lkif and.lkif are not necessarily identical.

However, the rules for import and export are sufficient to achieve at least the following goal: When exporting a tree A as a file a.lkif, and when importing that file back as a tree B, the trees A and B are equivalent as far as their exportable aspects are concerned. (Future improvements to the translator might make it necessary to reconsider this goal to some extent.)

3.3 RuleWise

RuleWise uses an UML/OCL representation formalism that has close links with what OWL provides, so the natural LKIF formalism to translate to is OWL-DL.

From the literature we learn that there are several approaches possible to translate UML to OWL:

a. Using a direct coupling between an UML modelling tool and an OWL modelling tool;
b. Making a UML meta-model for the RuleWise language and translating that to an OWL model
c. Making an XSLT that provides a mapping between the RuleWise model and LKIF

On the market more and more tools become available that provide a direct coupling between UML and OWL. This is logical since both languages are suited for modelling conceptual layers. A tool that can import UML models for instance is Topbraid Composer. It imports an UML2 model and transforms it into OWL. RuleWise uses UML, but has also extended UML on a couple of points. For instance for references a special kind of class (Package Reference) is used and traceability from model elements to the source is kept by using a special kind of traceability relation. These things reflect essential elements of law, and are therefore important elements in the RuleWise model. Using a coupling between standard UML and OWL would result in losing these special features, and therefore this approach has not been chosen for Estrella.
Translating an UML model in OWL via meta-modelling (eventually via MOF, Meta Object Facility, provided by the OMG group) is probably the most fundamental

approach. Although the results of this approach seem promising (see [1] and [2]), this is still a relatively unexplored road with many uncertainties underway and no secure results. Also in the literature there seems no agreement on the best approach.
In the end we decided on the third approach, making a XSLT that provides the mapping, as the preferred approach. In this manner we could keep the RuleWise specific features and we can be very certain that results in a practical usable solution.

4 Outputs

4.1 RuleBurst

```
<lkif about="KT" xmlns:lkif="http://www.w3.org/2000/01/lkif-schema#"
xmlns:egna="http://www.ruleburst.com/studio/interchange"
xmlns:lmth="http://www.w3.org/1999/xhtml">
  <s id="b1" summary="intermediate clause 1"></s>
```

These “s” tags define the statements which are referred by ID below in the premise, exception, and conclusion tags. For each statement used below, there will be a tag with a matching id here, to show the natural language statement referenced.

```
  <argument-graph>
```

This is the main argument graph, composed of “argument” tags for which the “direction” attribute is either pro or con (Negated). Each Argument represents a separate rule in RuleBurst, any contains any items which are identified to be important to the outcome of the Rule. It should be noted that “Intermediate Premises” will not be included in the export to LKIF.

```
    <argument direction="pro" id="b14@Rules_Rules_doc">
      <premise statement="b1" polarity="negative"></premise>
      <premise statement="b4"></premise>
```

Each argument is composed of a list of premises (Which can have a “polarity” attribute set to negative) and exceptions, ending with a single conclusion. The “statement” attribute is a reference to one of the “s” tags above. In an exported LKIF document, only premises and a conclusion will be present, with a premise for each Boolean attribute, or variable statement, in the original RuleBurst Document.

```
      <conclusion statement="b14"></conclusion>
    </argument>
  </argument-graph>
</lkif>
```

The concept of disjunction in a RuleBurst document is exported out to LKIF as multiple arguments, since there is no way to represent this in a single argument. This unfortunately is not always successful, especially when disjunction and conjunction are mixed in a single rule. As such it is important that the Rule Developer be aware of

[1] Brockmans, S., Colomb, R.M., Haase, P., Kendall, E.F., Wallace, E.K., Welty, C. and G.T. Xie; A model driven approach for building OWL DL and OWL Full Ontologies, in: Cruz, I.; Decker, S.; Allemang, D.; Preist, C.; Schwabe, D.; Mika, P.; Uschold, M.; Aroyo, L. (Eds.); Proceedings of the International Semantic Web Conference, 2006.

[2] Gašević, D.V., Djurić, D.O. and V.B. Devedžić, Bridging MDA and OWL ontologies, in: Journal of Web Engineering, Vol 4, No. 2, 2005, pp118-143.

this and make appropriate modifications either to the original rules, or the subsequent LKIF document.

4.2 KnowledgeTools

When translating from knowledgeTools to LKIF, the LKIF model contains a statement for each node of the knowledgeTree.
(Currently, only one statement is generated for each node. Future versions of the translator might map some nodes to multiple statements, one for each path the node is being used in.)

As mandated by LKIF schema, every statement has an ID, which is referenced by premises and conclusions in the following argument graph.

The argument graph is built from nodes that have children. (Leaf nodes are not currently mapped to arguments. Future version of the translator might also translate knowledgetools processes, and would add an argument for each leaf node that has has been entered as true or false in that process.)

knowledgeTools nodes for "and" and "not-or" are translated into a single argument, with one premises for each child node. knowledgeTools nodes for "or" and "not-and" are translated into one argument for each child node, with one premise each referring to that child node.

A polarity attribute with value "negative" is used to implement "not-and" and "not-or".

knowledgeTools does not currently generate LKIF models using direction attributes of value "con" (but might do so in the future). Other LKIF features that are not needed for the translation from knowledgeTools include "assertion", "exception", and "rule" elements.

Conceptually, the translation from LKIF to knowledgeTools is a two-step process. First, the file that is to be translated is simplified. In a second step, the simplified XML is imported.

The simplification step replaces elements as follows:

- Each <assumption> element is replaced using a <premise> element carrying the same attributes.
- Each <exception> element is replaced using a <premise> element carrying the same attribute value for “statement” and the opposite value for “polarity”.
- Each <argument>-Element carrying an attribute “direction=con” is replaced using an <argument> without attributes and with the following children: For each child <premise> of the original argument, a copy of that child is added with the opposite value for “polarity”. An unchanged copy of the <conclusion> child is added.

In the following import step, a node is created for each statement in the simplified XML, using its "summary" attribute as the node title. The function and children of the node are determined as follows:

1. If there is no argument with this statement as a conclusion, the node has no children.
2. If there is exactly one such argument:
   - If all premise elements in that argument have negative polarity, function "nor-or" is used. Each premise is added as a child.
   - Else, function "and" is used. For each premise, add a child that is computed depending on the polarity. If the polarity is negative, add a blank node with "not-or" function.; add the premise as a child to the blank node and use the blank node as a child to the current node. If polarity is not negative, use the premise as a child.

1. If there is more than one such argument:
   - If every premise in such an argument has negative polarity, "not-and" is used. For each argument, add a child that is computed depending on its premises. If the argument has exactly one premise, add that as a child. Otherwise add a blank node with "or" function; add all premises as children to the blank node and use the blank node as a child to the current node.
   - "or" is used. For each argument, add a child that is computed depending on its premises. If the argument has exactly one premise, and that premise does not have negative polarity, use that premise as a child. If the argument has exactly one premise, and that premise has negative polarity, add a blank node with "not-or" function; add the premise as a child to the blank node; use the blank node as a child to the current node. If the argument has more than more premise, create a blank node and use it as a child; compute the blank node's children as described above for "and" nodes.

As a start node, either use a statement selected by the user or use the statement indicated by the "issue" attribute in the argument-graph element.

4.3 RuleWise

Two translators were build, one that translates a RuleWise model into an LKIF OWL model and one that translates an LKIF OWL model in to a RuleWise model.

From RuleWise to OWL

The mapping from RuleWise elements to OWL elements that is expressed in the XSLT as it is now is shown in the diagram below:

RuleWise (UML) model	*LKIF (OWL) model*
Model	Ontology
Package	Package (individual)
Class	Class
Operation (used for attributes in RuleWise tool)	DatatypeProperty (with a specific class range)

Associations	ObjectProperties
Generalization	Sub-class relation
Constraint (comment form)	Comment

Some additional remarks are:

- an owl:Ontology contains at least one individual: the model, which has the type 'Model' as defined in the rulewise ontology.
- UML:Namespace.ownedElement is translated into rw:ownedElement define relations between packages (and models), and classes.

From OWL to RuleWise

The mapping between OWL and RuleWise as expressed in the XSLT is again showed in the table:

LKIF (OWL) model	*RuleWise (UML) model*
Model	Model
Package	Package
Class	Class
ObjectProperties with inverseOf	Associations

The Link from packages to model elements can be found in rw:ownedElement, this is translated into UML:Namespace.ownedElement

Since OWL acknowledges more independent elements than UML (e.g. properties are independent in OWL, whilst in UML they exist within the context of a class) for attributes and sub-classes the translation back into OWL is not trivial. More research into this is needed and the XSLT must be adjusted on the basis of the result of this research.

5 Testing

5.1 RuleBurst

Four tests have been performed on the resulting Compact XML produced by the RuleBurst to LKIF translators.

- **Validation against the LKIF XSD**
  - LKIF documents produced by the parser were first validated against the XSD to ensure they conformed. This check was performed every time a change was made to the translator to ensure that any changes made did not break validation
- **Manual Testing of Resulting Output**
  - Once the documents had been confirmed as valid LKIF they were then checked over manually, as a way of attempting to confirm the Rule Structure had been maintained. This check was performed by both the

Technical and Rule orientated developers involved in production of the translator.

- **Round trip testing RB > LKIF > RB**
  - The rules were imported back into RuleBurst from the LKIF document, and the resulting RuleBurst project was then compared to the original source project to check for inaccuracies. Tracing back through the stages of translation provided a good way of checking where changes to Rule Structure (If any) occurred.
- **Round trip testing RB > LKIF > KT > LKIF > RB**
  - The full round circle test was performed, starting and terminating at both KT and RuleBurst. This was mainly a test that the individual interpretations of the LKIF structure by RuleBurst and KnowledgeTools were the same, but also provided a final sanity check on the translators.

5.2 KnowledgeTools

We tested both the import from LKIF to knowledgeTools and the export from knowledgeTools to LKIF.

For the export we used the knowledgetree: “Taxation directive” (the knowledgeTools´ pilot). Here we could successfully generate a corresponding xml-file (LKIF). We then tried to re-import this xml-file to knowledgeTools and here, too, we succeeded. As a result we could assert, that the re-imported tree equalled the original tree and that there was no loss of information.

For the import we tried two different xml-files:

- The Hungarian pilot (vatFull-argumentGraph.xml)
- A file from Ruleburst (Transformed.xml)

Both files helped illustrate differences between our translators:

- The knowledgeTools translator did not have support for arguments without premises (which imply that their conclusion is always true). Instead, blank nodes were created in this situation. Future versions of the translator will at least avoid blank nodes, and possibly also create knowledgeTools „processes“ to preserve these arguments.
- The file we received used src attributes to identify statements, while knowledgeTools expects statements to be identified using XML IDs.

5.3 RuleWise

After testing with a small example from the Dutch Social Security domain, real testing was done on the model of the EU directive on mergers for pilot three. The results of these tests are reported on in D2.5 and D5.4.

6 Conclusions

6.1 RuleBurst

At present the RuleBurst translator is in a state that showcases all the vital sections of the Vendor to LKIF journey. Whilst there are certain limitations, none of these impede the immediate goal of the translator as a demonstrative tool for the review. Further develop is necessary, but this is on an enhancement basis as an attempt to integrate more of the features supported by RuleBurst into LKIF, either in it's existing form, or with modifications.

Some modifications may need to be done to ensure that the Translator uses the most precise method of conversion possible, but its current form is a more than apt proof of concept. With full round trip capabilities also available, both to and from LKIF, and also through KT, the translator is a solid base for further refinement.

Progress is also being made towards producing an Exporter which supports the LKIF Rule Syntax. This, along with enhancements to the LKIF standard itself, have the potential to allow support for not only Variables, but also entities. At the present moment in time this has not been implemented, but the conceptual design seems possible.

6.2 KnowledgeTools

The representation of argument graphs in LKIF is flexible enough to describe knowledgeTrees. Converting back from general LKIF argument graphs to a form that matches knowledgeTrees is somewhat more challenging.

Through implementation of the translator and thanks to feedback we received, we have identified several issues in which the translation mechanism could possibly be improved in the future, including the following tasks:

- Translation of LKIF to knowledgeTools processes. LKIF files are currently translated to knowledgeTrees, which cannot include information about arguments without premises. Instead, it would be possible to translate LKIF files to pairs of trees and „processes". Conversely, a translation of trees and processes back into LKIF could be interesting.
- Unique nodes. A full translation from LKIF to knowledgeTools can sometimes require a feature called „unique" nodes in D 1.2. While unique nodes are still intended to be used for this purpose, they have not been implemented yet.
- Conversely, future versions of the translator from knowledgeTools to LKIF could translate „path-dependent" nodes into more than one statement. An implementation this change would be possible before „unique" nodes are available, but makes it harder to ensure that a round-trip (the translation of a tree to LKIF and back) would result in the original tree.
- In a tree with an „and" node (translated to a statement „parent") that has two children (translated to statements „child1" and „child2"), the translator currently generates LKIF containing the argument that „parent" follows from the premises „child1" and „child2". Future versions of the translator might also add an argument that „parent" is false if either of „child1" and „child2" are false. If this interpretation is implemented, imports from LKIF to

knowledgeTools should ideally recognize this situation and avoid the translation to multiple nodes for these two arguments, collapsing them back into one node. The interpretation is LKIF files that only contain one of these two arguments is an open question.

- Since implementation of the translator, a possible translation for „xor“ nodes has been suggested, which might be added to future versions of the translator.

Note that the current implementation of the translator already covers a large subset of LKIF features as well as knowledgeTools features, and a completion of research and implementation of the tasks listed above is not necessary for use of the translator in most situations.

Some of the tasks and issues listed above are revelant only depending on knowledgeTools features that might or might not be implemented in knowledgeTools in the near future. Only if and when said features are implemented, improvements to the translator might also be possible.
At this point, knowledgeTools is not committed to a roadmap for the implementation of said features.