Using a Task-Oriented Framework to Characterize Visualization Approaches

Marcelo Schots, Cláudia Werner
Systems Engineering and Computing Program – Federal University of Rio de Janeiro (PESC/COPPE/UFRJ)
Rio de Janeiro, Brazil
schots@cos.ufrj.br, werner@cos.ufrj.br

***Abstract*— Visualization approaches support stakeholders in a variety of tasks. However, they are spread in the literature and their information is usually not clearly organized, classified and categorized, which makes them hard to be found and used in practice. This paper presents the use of a task-oriented framework in the context of a characterization study of visualizations that provide support for software reuse tasks. Such framework was extended in order to capture more detailed information that may be useful for assessing the suitability of a particular visualization. Besides enabling a better organization of the findings, the use of the extended framework allows to identify aspects that lack more support, indicating opportunities for researchers on software reuse and software visualization.**

***Index Terms*—Software visualization, knowledge structuring, taxonomy framework, software reuse.**

I. Introduction

Research on software visualization has been increasingly growing in the last years, and there are several approaches available nowadays for supporting the most diverse software development tasks. While this is a good outcome, a side effect is that it becomes harder to find or choose the "right" visualization for supporting a particular task. Given that no single visualization can address all software engineering tasks simultaneously, it becomes imperative to organize and classify existing approaches.

This can be done, for instance, through taxonomies. They are useful for the scientific community to get an overview of existing approaches, how they compare to each other, and what gaps or challenges for future research exist. An example of a taxonomy framework is the one proposed by Maletic et al. [1], which presents a task-oriented view of software visualizations through dimensions that highlight the strengths of individual tools and techniques regarding their application to software engineering tasks, providing a space of possible visualization systems with respect to such tasks [1].

Although this framework has been used in some works for classifying their own visualization tools, we did not identify any comprehensive research so far that used it for characterizing and categorizing existing visualizations focusing on a particular software engineering field of interest. In fact, the broader classification with this framework was done by the authors themselves, presenting examples of selected visualizations mapped along the proposed dimensions [1].

A drawback of this framework is that there is no support on how to map features of existing visualizations to the proposed dimensions, which makes it harder to perform an objective classification. Moreover, two important aspects are not clearly emphasized in its dimensions: the *requirements* of the visualization tools and *evidence* on their applicability or viability. The lack of such information can hinder the choice of visualizations and make potential users reluctant in using them.

In this paper, we present the use of the task-oriented framework for organizing the findings of a secondary study to characterize visualization approaches that support software reuse tasks. To this end, we extended it by adding two complementary dimensions and including supplementary support for mapping information to the dimensions, through a set of questions and fields that aim at aiding the categorization.

II. Overview of Maletic et al.'s Framework

The task-oriented framework proposed by Maletic et al. takes into account previous work on taxonomic descriptions for emphasizing general tasks of understanding and analysis during the development and maintenance of large-scale software systems [1]. Its dimensions reflect the why, who, where, what, and how of the software visualization, as follows:

Task: A visualization system aims at supporting the understanding of one or more aspects of a software system, and this understanding process will in turn support a particular task [1]. Thus, this dimension indicates what particular software engineering tasks are supported by the visualization [1].

Audience: This defines the attributes of the users of the visualization system [1]. Besides being oriented to distinct roles, different tools can also be tailored towards users with different skills (e.g., experienced versus beginner, developer versus manager etc.). An experienced developer may have different information needs than a novice team member [1].

Target: The target of a software visualization system defines which (low level) aspects of the software are visualized, i.e., the work product, artifact, or part of the environment of the software system [1]. Examples include architecture, design, algorithm, source code etc. Other types of target are software metrics, process information, and documentation; this can support the software process and team management activities [1]. Software development surroundings also provide several aspects that can be visualized.

Representation: This dimension shows how a visualization is constructed based on the available information [1]. An aspect on which the effectiveness of information visualization hinges is its ability to clearly and accurately represent information. The relationship between data values and visual parameters must be univocal; otherwise, it may not be possible to distinguish one value's influence from the other [1].

Medium: The effectiveness of visualizations also relies on humans' ability to interact with them to figure out what the information means. The medium is where the visualization is rendered, i.e., some display technology from which the user interacts and perceives the visualization [1]. The medium dictates how interactions may occur; each one has different characteristics and hence is suited for different tasks [1].

III. Extending the Framework Dimensions

In order to complement the framework with information that is relevant to the visualization users, we propose two additional dimensions that are not (or at least not directly) addressed in the original framework: one related to the **requirements** of the visualization approaches (***which***) and other related to **evidence** on their use (***worthwhile***), as follows.

A. Requirements – which resources are required/used in the visualizations?

Although the original framework dimensions provide an organization of the goals and concepts implemented in the visualizations, it is not possible to distinguish what is needed to deploy and execute the tools. For instance, an important concern in any interactive visualization system is performance, in particular responsiveness to changes triggered by their direct manipulation. Certain visualizations may be costly, not only in terms of processing but also due to the specific hardware and software solutions on which they depend. This cost can be expressed (to some extent) in terms of the visualizations' hardware and software requirements, besides the programming languages, APIs and frameworks reused for building it.

Visualization requirements can also provide indication on potentially conflicting configurations, e.g., if a given version of a framework used for building the visualization is not compatible (or has known behavior issues) with a certain hardware or software used by the organization. The prevention of conflicts can avoid unnecessary waste of time.

B. Evidence – are the proposed visualizations worthwhile?

In order to determine if visualizations are effective in helping their target users, it is desirable to expose them to a proper evaluation [2]. The lack of evaluations is a shortcoming not only of software visualization research, but of software engineering and computer science in general. In fact, many developers perform very limited or no evaluation at all of their visualization tools [2], making their effectiveness unclear.

This aspect is not emphasized in the original framework; quality attributes that illustrate the usefulness of the approaches (e.g., effectiveness) are put into the *Representation* dimension. This may "obfuscate" their importance and decrease their visibility. Thus, this dimension aims at characterizing what kinds of evaluations and assessments were carried out with the visualization (if any), with the identified limitations in its use, providing insights of the visualizations' worthiness beforehand.

IV. Structuring the Mapping to the Dimensions

As previously mentioned, the original framework does not provide support on how to map features and aspects of existing visualizations to the proposed dimensions. Capturing details in a lower-level can ease to find relevant information and map it into these dimensions. For instance, for the *Task* dimension, it is important to identify which problems, motivations or issues leaded to the development of each system, in order to recognize why each of the visualizations is needed in a given scenario.

In this sense, we suggest a more detailed structuring of each dimension by means of questions that help depicting them, as well as information related to these questions (shown in Table I). Part of this structuring was based on the original framework, and additional fields were included from the needs identified for organizing the findings of the secondary study.

In order to show an example of use of the extended taxonomy, the following sections discuss this study and its findings. The detailed use of the framework is presented in [3].

V. Using the Framework in a Secondary Study

We demonstrate the use of this extended framework in practice by presenting its application for better organizing the findings of a secondary study [3] aimed at characterizing and identifying visualization approaches that can be used for supporting software reuse, regardless of the focus of support. The primary research question (*PQ*) of the study is: *Which visualization approaches have been proposed to support software reuse?* This question was decomposed into secondary (*SQ*) and tertiary (*TQ*) questions, as indicated in Table I.

In total, 36 publications describing 34 approaches were selected. For illustration purposes, some of them are mentioned throughout the description of the findings. The full list of publications and additional details (e.g., the search string used, the publication selection procedures, and limitations of the study) can be found in [3]. For each dimension, the selected publications were the only sources of information.

In the *Task* dimension, regarding how visualizations support software reuse (*SQ1*), there was already a concern on allowing reuse of a variety of artifacts since the first identified works were published (e.g., [4]). Approach goals are mostly artifact-oriented rather than process-oriented. Although approaches encompass many software engineering activities (*TQ1.1*), only a few of them present integration among activities. A variety of reuse tasks (*TQ1.2*) are supported, but understanding assets for reuse is by far the most common one.

In terms of the different aspects that can be the focus of comprehension, most approaches support the understanding of an asset's structure (e.g. [5]), and some help understanding their behavior (e.g. [6]). One work deals with evolution information. Other supported tasks include integrating reusable assets (10 approaches), searching and retrieving reusable assets (7), discovering and evaluating potentially reusable assets (5), and restructuring assets for reuse (2). Nevertheless, there is few integrated support for such tasks.

TABLE I. SOFTWARE VISUALIZATION DIMENSIONS, QUESTIONS AND ASSOCIATED INFORMATION

Dimensions	Questions to be addressed	Associated information to be extracted
Task (why)	SQ1: How do visualizations support the software engineering field of interest?	Approach motivation/Assumptions
		Approach goals
		Visualizations' specific goals
	TQ1.1: Which software engineering activities are addressed by the visualizations?	Software engineering activities addressed by the visualizations
	TQ1.2: Which tasks are supported by these visualizations?	Tasks related to the field of interested that are supported by the visualizations
Audience (who)	SQ2: To which stakeholders are these visualizations intended/targeted?	Visualizations' audience (stakeholders who can benefit from the visualizations)
Target (what)	SQ3: Which items/data are visually represented?	Visualized items/data (what is visualized)
	TQ3.1: Where do these items/data come from?	Source of visualized items/data
	TQ3.2: How are these items/data collected?	Collection procedure/method of visualized items/data
Representation (how)	SQ4: Which visualization metaphors are used?	Visualization metaphors used (how it is visualized)
	TQ4.1: How are data mapped to the visualizations?	Data-to-visualization mapping (input/output)
	TQ4.2: Which visualization strategies and techniques are employed?	Visualization strategies and techniques
Medium (where)	SQ5: Where are the visualizations displayed?	Device and/or environment used for displaying the visualizations (where it is visualized)
	TQ5.1: Which resources can be used for interacting with the visualizations?	Resources used for interacting with the visualizations
Requirements (which)	SQ6: Which hardware/software resources are needed to deploy and execute the visualization tools?	Hardware and software requirements/dependencies
	TQ6.1: Which programming languages, APIs and frameworks are used?	Programming languages, APIs and frameworks used for building the visualization
Evidence (worthwhile)	SQ7: Which methods are used for assessing the quality[a] of the visualizations (if any)?	Visualization evaluation methods
	TQ7.1: In which scenarios are the visualizations employed (if any)?	Application scenarios of the visualizations
	TQ7.2: Which aspects of the visualizations are evaluated (if any)?	Evaluated aspects
	TQ7.3: What are the results/outcomes of the conducted evaluations (if any)?	Visualization evaluation results/outcomes

a. Quality evaluation/assessment encompasses quality attributes such as effectiveness, efficacy, among others.

Regarding the *Audience* dimension, Fig. 1 (top) presents the proportion of stakeholders to which the visualization approaches are intended/targeted (*SQ2*). Whenever a publication seems to differ between common programmers and "reusers", it was decided not to merge these roles into a single one, as some approaches may have broader goals not related to reuse, and such goals may be indeed targeted to different roles.

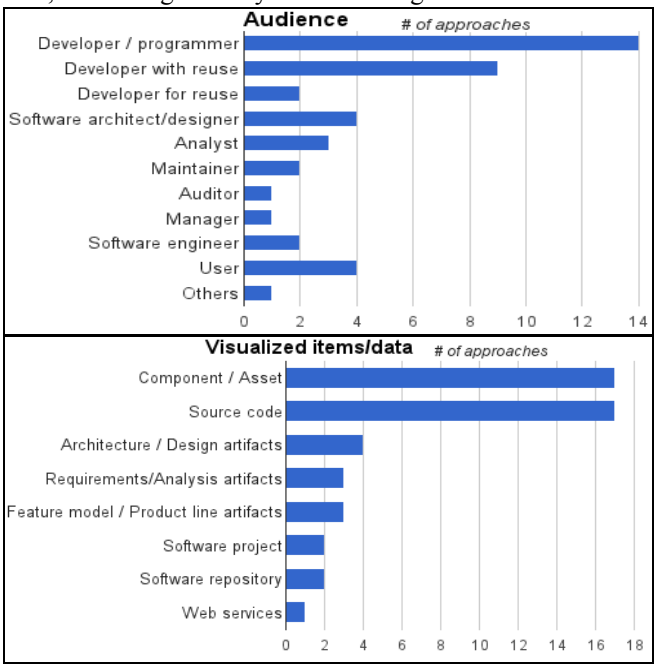

Fig. 1 . Visualization's audience (SQ2) (top) and visualized items/data (and related information) (SQ3) (bottom)

Programmers are the most supported stakeholders by far. 14 approaches mention programmers in general, while 9 support specifically developers with reuse (also referred to as "reusers" or "system integrators"). 2 approaches support developers for reuse (e.g., [6]). Other supported roles include software architects/designers, analysts, maintainers, and auditors. One approach mentions manager support ([4]), but it only presents low-level technical details on a software project.

Some approaches mention a more general role, such as software engineer or user (e.g., [7], [8]). One approach aims to support safety domain experts [9], not related to software development. In spite of these different profiles, only a few works support multiple stakeholders simultaneously.

In the *Target* dimension, Fig. 1 (bottom) shows which items/data are visually represented. The vast majority of the visualized items/data (*SQ3*) are source code entities (e.g., classes and relationships, or software components), but other kinds of artifacts from other software development stages are also visualized, as shown in the figure.

It is interesting to observe that the data sources (*TQ3.1*) are usually the source code of a program and databases. Only a few approaches combine information from different sources, and some are compatible with a limited set of data types. Common data sources that could provide reuse information are not explored (e.g., version control system repositories, issue trackers etc.). Moreover, although many assets have related data available online, such data are usually underexplored.

Since each visualization technique may impose constraints to their use, each collection procedure (*TQ3.2*) must make the proper arrangements. Some works also defend the use of

intermediary formats for storing the collected data (e.g., [7], [10]) in order to make them reusable in different visualizations.

In the *Representation* dimension, regarding the representation of data (*SQ4*), approaches show a major adoption of the network/graph and hierarchical metaphors – tree representations are the most common ones, e.g., hyperbolic tree (e.g., [7]), sideways tree, tree list (e.g., [5]), and interactive cone tree. Other types of abstractions are also used. Publications lack a discussion on how/why a given metaphor was chosen and whether it is effective or not in its purpose.

Table II shows a mapping between approaches and these graphical representation resources as described in the publications (more detailed information, including the shortcuts, can be found in [3]). The mapping between data and visualizations (*TQ4.1*) is barely described in most of the publications. In some cases, the reader/user has to "guess" it, which can be risky and lead to wrong interpretations of data.

TABLE II. DATA-TO-VISUALIZATION MAPPING (TQ4.1)

	COLOR	NODES AND EDGES/ARROWS	POSITION	SIZE/DIMENSION	BRIGHTNESS/CONTRAST	ORDERING	CONTAINMENT	ICON	DENSITY	GENERAL/CUSTOMIZABLE	N/A	TOTAL
[Mancoridis199374]		■			■			■	■			4
[Constantopoulos19951]	■	■										2
[Lange1995342]		■	■	■		■						4
[Helfman199631]			■		■							2
[Alonso1998483]										■		1
[Biddle199992]	■	■	■	■								4
[Biddle199992 / Marshall2001 / Marshall2001103]										■		1
[Ye2000266]			■					■				2
[Marshall2001103]											■	1
[Marshall2001103 / Anslow2004]										■		1
[Mittermeir200195]	■	■										2
[Charters2002765]	■		■	■				■				4
[Marshall200381]											■	1
[Anslow2004 / Marshall200435]											■	1
[Wahid2004414]	■											1
[Kelleher200550]		■										1
[McGavin2006153]	■	■	■		■							4
[Tangsripairoj2006283]			■	■								2
[Washizaki20061222]	■	■		■								3
[Gonçalves2007872 / Oliveira2007461]											■	1
[Holmes2007100]	■				■		■					3
[Stollberg2007236]		■	■									2
[Dietrich200891]		■					■					2
[Ali200950]	■		■	■		■	■					5
[Damaeviius2009507]			■									1
[DeBoer200951]	■											1
[López20091198]							■					1
[Anquetil2010427]	■	■	■									3
[Areeprayolkij2010208]		■										1
[Apel2011421]	■	■	■	■								4
[Duszynski2011303 / Duszynski201237]	■			■								2
[Bauer2012435]	■		■	■		■						4
[Feigenspan20121]	■			■		■						3
[Yazdanshenas2012143]	■											1
TOTAL	16	13	13	10	4	4	4	3	1	3	4	

Colors are the most frequent visual resource to represent variations of data. Nodes and arrows usually map to concrete artifacts and their relations, respectively. Position (along an axis) and dimensions (e.g., height) represent a given aspect of software; for instance, [11] uses the width of colored bars to indicate the total number of API calls. Some approaches offer a custom mapping between visual elements and data (e.g., [7]).

Although several visualization strategies and techniques are used (*TQ4.2*), only a few approaches make a comprehensive use of them. Some approaches may require more interaction resources for increasing their usability, and some visualization facilities are underexplored.

For instance, the most frequent zoom interaction is the geometric zooming (i.e., enlarging objects while zooming in and shrinking them while zooming out). In contrast, semantic zooming (showing different visual representations to information items according to the available space based on zooming interactions) was only found in one approach. Filtering, in turn, is applied by collapsing/expanding a set of elements (e.g., [5], [11]), including/removing an element from the view (e.g., [4]), highlighting items of interest (e.g., [9]) or tuning/tweaking (through parameters customization). Some approaches offer more than one kind of filtering (e.g., [6]).

There is a lack of mechanisms that offer flexibility to software stakeholders in customizing their visualizations. Some works try to generate flexible visualizations, but they usually require expertise knowledge (e.g., programming skills for configuring/mapping views and data) to such customization.

Many publications (12) do not specify information about the *Medium* dimension (*SQ5*). One approach [12] uses a virtual reality environment (VRML-enabled browser or standalone viewer) for displaying information (not specifying the physical medium). The other 33 approaches present visual information in a computer screen. From these, 10 contain information associated with the environment: 6 use the web (e.g., [7]) and 4 employ Eclipse (e.g., [5]) or an extension of it. The remaining 23 approaches are standalone tools or characterize their own environment, but only around half of them (11) make such information explicit. No approach mentions mobile devices.

Regarding the resources used for interacting with the visualizations (*TQ5.1*), only 2 approaches explicitly state the use of both keyboard and mouse, while other 13 approaches mention mouse interactions. Among the remaining approaches, it is assumed that 20 provide mouse support and 3 support keyboard, 2 of these supporting both devices. As it can be seen, visualization systems still largely focus on these peripherals for interacting with data. The approach that runs in a virtual reality environment does not mention interaction resources.

In the *Requirements* dimension, in order to deploy and execute the systems and their visualizations (*SQ6*), 4 approaches require Eclipse IDE (e.g., [5], [8]), and other 2 require Jun for Java (one of these also requires OpenGL). In general, approaches lack integration with development environments, which hampers communication with other tools. In terms of hardware, no special requirements are mentioned.

The selected approaches use a variety of programming languages, APIs and frameworks (*TQ6.1*), being Java the most adopted language, explicitly mentioned by 16 approaches. For building the visualizations, Prefuse is used by 3 approaches. Other visualization frameworks include the Tk graphics library (Tcl/Tk), JPowerGraph, Grappa and JUNG.

Software and hardware requirements are not well discussed in many publications, hampering a proper evaluation of the feasibility of the approaches to particular contexts. The same occurs with programming languages, APIs and frameworks.

In the *Evidence* dimension, regarding the methods used for assessing the quality of the visualizations (*SQ7*), almost half of

the approaches (16 out of the 34) do not have any evaluation of their use. In some cases, at most, a simple example of use is shown. This is partially explained by the lack of demand for evidence in publications (a scenario that has changed in the last years). The absence of proper evaluations may raise questions as regards to meeting the purpose to which they were proposed.

Among the remaining 18 approaches, one publication presents a semi-controlled experiment [5] and another one presents 3 experiments. The other 16 are evaluated in terms of their use in practice, and 5 of them mention external subjects (e.g., [9]), applying user tests and a survey or exploratory studies and a structured interview [9]. Only a few of the other 11 evaluations in practice specify that they were performed by the authors themselves (e.g., [8]), but we believe they all fit into this setting, since no external subjects is mentioned.

Regarding the scenarios in which the visualizations are employed (*TQ7.1*), from the 18 approaches that present some kind of evaluation, 6 are performed in an academic context, 7 use open source data (which allows verification of results), and 9 show studies involving commercial projects (demonstrating interactions established with industry). Some approaches use more than one scenario. One approach does not specify the evaluation scenario. Fig. 2 summarizes such information.

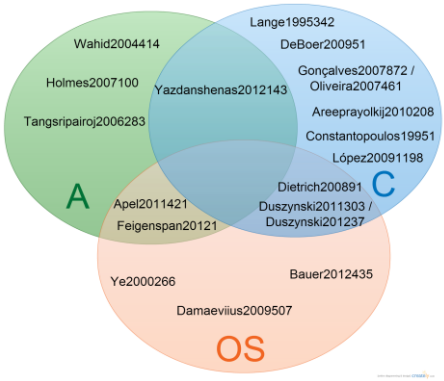

Fig. 2 . Evaluation scenarios (**A**cademic, **C**ommercial, **O**pen **S**ource) (TQ7.1)

In general, the reported data about the evaluations lack more details regarding scenarios in which they were conducted (*TQ7.1*), which aspects were evaluated and why (*TQ7.2*), how the analyses were made, and which strengths and opportunities for improvements were identified (*TQ7.3*).

VI. Final Remarks

The study presented in this paper demonstrates a concrete use of the taxonomy framework, and serves as a summarized catalog of visualization approaches geared to software reuse, whose further details can be obtained in [3] and in the corresponding original publications. The extended framework not only allows to organize the findings of the study in terms of visualization dimensions, but also highlights aspects that lack support, and may indicate research opportunities on software reuse and software visualization.

During this research, it was noticed that many publications lack a clear description of information related to these dimensions, especially regarding the *medium* for displaying visualizations, their hardware/software *requirements*, and any *evidence* on their use. The questions shown on Table 1 helped to search for and organize the dimensions information.

Some possibilities for future work include (i) an analysis on the connections between the dimensions and their attributes, (ii) further studies on software visualization applied to another field (e.g., software maintenance), and (iii) other framework extensions for encompassing additional aspects of interest.

Acknowledgment

The authors would like to thank CNPq, CAPES and FAPERJ for the financial support and the anonymous reviewers for their helpful comments and insights.

References

[1] J. I. Maletic, A. Marcus, M. L. Collard, “A Task Oriented View of Software Visualization,” in *1st Int’l Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, 2002, pp. 32–40.

[2] M. Sensalire, P. Ogao, A. Telea, “Evaluation of software visualization tools: Lessons learned,” in *5th Int’l Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, 2009, pp. 19–26.

[3] M. Schots, R. Vasconcelos, C. Werner, “A Quasi-Systematic Review on Software Visualization Approaches for Software Reuse,” Tech. Rep., Federal University of Rio de Janeiro, 2014.

[4] S. Mancoridis, R. C. Holt, D. A. Penny, “Conceptual framework for software development,” in *1993 ACM Computer Science Conference*, 1993, pp. 74–80.

[5] R. Holmes, R. Walker, “Task-specific source code dependency investigation,” in *4th Int’l Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, 2007, pp. 100–107.

[6] R. Biddle, S. Marshall, J. Miller-Williams, E. Tempero, “Reuse of Debuggers for Visualization of Reuse,” in *1999 Symposium on Software Reusability (SSR)*, 1999, pp. 92–100.

[7] O. Alonso, W. Frakes, “Visualization of Reusable Software Assets,” in *6th Int’l Conference on Software Reuse (ICSR)*, 2000, pp. 251–265.

[8] S. Duszynski, J. Knodel, M. Becker, “Analyzing the source code of multiple software variants for reuse potential,” in *18th Working Conference on Reverse Engineering (WCRE)*, 2011, pp. 303–307.

[9] A. R. Yazdanshenas, L. Moonen, “Tracking and visualizing information flow in component-based systems,” in *20th Int’l Conference on Program Comprehension (ICPC)*, 2012, pp. 143–152.

[10] C. Anslow, S. Marshall, J. Noble, R. Biddle, “Software Visualization Tools for Component Reuse,” in *2nd Workshop on Method Engineering for Object-Oriented and Component-Based Development, 19th OOPSLA*, 2004.

[11] V. Bauer, L. Heinemann, “Understanding API usage to support informed decision making in software maintenance,” in *16th European Conference on Software Maintenance and Reengineering (CSMR)*, 2012, pp. 435–440.

[12] S. M. Charters, C. Knight, N. Thomas, M. Munro, “Visualisation for informed decision making; from code to components,” in *14th Int’l Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2002, pp. 765–772.