PROCEEDINGS

SEKE 2009

The 21st International Conference on Software Engineering & Knowledge Engineering

Sponsored by
Knowledge Systems Institute Graduate School, USA

Technical Program
July 1-3, 2009
Hyatt Harborside Hotel, Boston, Massachusetts, USA

Organized by
Knowledge Systems Institute Graduate School
Table of Contents

Foreword ........................................................................................................ iii

Conference Organization ........................................................................ iv

Wireless Computing, Networking and Sensing

Dr. H. T. Kung ............................................................................................. 1

Virtual Spaces: From the Past to the Future

Dr. Shi-Kuo Chang ....................................................................................... 2

Software Engineering of Autonomic Grid Computing Systems and Applications

Web Services Reliability Patterns (S)
Ingrid Buckley, Eduardo B. Fernandez, Gustavo Rossi, S. Masoud Sadjadi ............... 4

Consistency in Self-Reconfiguration of Self-Healing Systems
Michael E. Shin, Kiran Gopal Reddy Sunanda .............................................. 10

Task Decomposition for Adaptive Data Staging in Workflows for Distributed Environments (S)
Onyeka Ezennwye, Balaji Viswanathan, S. Masoud Sadjadi, Liana Fong, Gargi Dasgupta, Selim Kalayci .............................................................. 16

Requirements

Constructing FODA Feature Diagrams with a GUI-based Tool (S)
Shin Nakajima .............................................................................................. 20
Decision Support System Environment for Software Architecture Style Selection (DESAS v1.0) (S)
Shahroz Mozaven, Hamed Akjadi, Jafar Habibi, Ali Kamandi ........................................ 147

Towards Architecture-centric Collaborative Software Development (S)
Yun Chun Sun, Hai Song, Wenpin Jiao ........................................................................ 152

Agents and Multi-Agent Systems

Analysis of Agent Oriented Software Engineering Methodologies for Social Causal Models
Michele Atkinson, Sheryl Duggins ............................................................................ 157

Realization of Semantic Search Using Concept Learning and Document Annotation Agents
Behrouz H. Far, Cheng Zhong, Zilan Yang, Mohsen Afsharchi .................................. 164

Agent-based Simulation Model for the Evolution Process of Open Source Software
Taemin Seo, Heesong Lee ......................................................................................... 170

Towards Merging Goal Models of Networked Software
Zaiwen Peng, Keqin He, Rong Peng, Jian Wang, Yutao Ma ...................................... 178

Comparison of Some Single-agent and Multi-agent Information Filtering Systems on a Benchmark Text Data Set (S)
Snehis Mukhopadhyay, Shengquan Peng, Rajeev Raje, Mathew Palakal, Javed Mostafa ...... 185

Towards Adaptable BDI Agent: A Formal Aspect-oriented Modeling Approach (S)
Lily Chang, Xudong He ............................................................................................ 189

A Multi-agent Debugging Extension Architecture (S)
Ziad Al-Sharif, Clinton Jeffery .................................................................................. 194

A Recognition-primed Architecture for Human-centric Multi-agent Systems
Xiaocong Fan ........................................................................................................... 200

Using Knowledge Objects to Exchange Knowledge in a MAS Platform
Ana Paula Lemke, Marcelo Blois .............................................................................. 206

JAAF: A Framework to Implement Self-adaptive Agents
Baldino F. dos S. Neto, Andrew D. da Costa, Manoel T. de A. Neto, Viviane T. da Silva, Carlos J. P. de Lucena ........................................... 212

An Agent-based Centralized e-Marketplace in a Virtual Environment (S)
Ingo Seidel, Markus Gartner, Josef Froschauer, Helmut Berger, Dieter Merkl ............... 218

Interoperability and Semantic Web Technologies

Semantic Service Matchmaking in the ATM Domain Considering Infrastructure Capability Constraints (S)
Thomas Moser, Richard Mordini, Wikan Danar Sunindyo, Stefan Biffl ........................ 222

Ontology Mapping Representations: A Pragmatic Evaluation (S)
Hendrik Thomas, Declan O'Sullivan, Rob Brennan ................................................. 228

Bridging Semantic Gaps Between Stakeholders in the Production Automation Domain with Ontology Areas
Stefan Biffl, Wikan Danar Sunindyo, Thomas Moser .............................................. 233

LD2SD: Linked Data Driven Software Development
Aftab Iqbal, Oana Ureche, Michael Hausenblas, Giovanni Tummarello ....................... 240

Improving Searchability of a Music Digital Library with Semantic Web Technologies
Paloma de Juan, Carlos A. Iglesias ......................................................................... 246

A Guideline Engine For Knowledge Management in Clinical Decision Support Systems (CDSSs)
Michele Czeccharelli, Alessandro De Stasio, Antonio Donatiello, Dante Vitali ............ 252

Ontology-based Semantic Annotations of Medical Articles (S)
Jihen Majdoubi, Mohamed Tmar, Faiez Gargouri .................................................... 258
The reasons for this are the multitude of mapping/matching approaches available, e.g. different matching algorithms, matching types etc. This approach has unique requirements for mapping representation, simply because different information and structures need to be represented to express a correspondence. The design of a mapping representation which fulfills all those requirements might be too complex or could lead to a format which represents only the smallest common denominator. For example INRIA is generic but, compared to proprietary formats e.g. XML, detailed Multiple mapping representations may be unavoidable because for different mapping scenarios, different representations of the mapping information are necessary. The correct data which documents the mapping lifecycle is more uniform and for most cases it is beneficial to develop the concept of a flexible enrichment of existing and future ontolo- gy mapping representations in order to augment their usage, reuse and management. In particular, in an ontology based meta-layer a common vocabulary for modelling life-cycle meta-data could be established and limited to the individual formats representing mapping correlations [20]. Established mapping formats and tools don’t need to be changed but available meta-data can still be stored and retrieved in a structured, documented and predictable way.

In conclusion, the remarkable efforts to support the creation of ontologies as mappings are just the first step. Further research is needed to develop more powerful concepts for the management, sharing and reuse of ontology maps- pings to support the flexible communication of a common understanding which is large enough to control the overall information glint [1].

ACKNOWLEDGMENTS

This work is partially funded through the Science Foundation Ireland FAME project (award No. 08/SRC/16168).

REFERENCES


A. Integration of Heterogeneous Systems

System integration is the task to combine a range of smaller systems to appear as one big system. There are several levels at which system integration could be performed [3], but there is so far no standardized integration process that explains how to integrate systems in general.

Typical integration solutions focus either on technical heterogeneity (how to connect systems that use different platforms or protocols) or on semantic heterogeneity (how to translate data in messages between systems that use different data models or languages). In order to cope with both types of heterogeneity, system integration approaches at service level middleware technology [9] supports syntactical transformation between services, while the semantic heterogeneity of services can be addressed with a common data schema [13]. Limitations of these integration approaches are: 1. The need for a common data schema [13], which is hard and time-consuming to negotiate, sometimes impossible if stakeholders continue to disagree. 2. The need for integration over heterogeneous middleware technologies (with different APIs or network architectures) implies the development of static and therefore inflexible wrappers between each combination of middleware technologies, and thus increases the complexity of communication.

Semantic integration is defined as the solving of problems originating from the intent to share data across disparate and semantically heterogeneous data [13]. These problems include the matching of ontologies or schemata, the detection of duplicate entries, the reconciliation of inconsistencies, and the modelling of complex relations from different sources [9]. Over the last years, semantic integration becomes increasingly crucial to a variety of information-processing applications and has received much attention [13].

B. Ontologies for Semantic Integration

An ontology is a representation vocabulary for a specific domain or subject matter, like production automation. More precisely, it is not the vocabulary as such that qualifies as an ontology but the concepts that the vocabulary is intended to capture [5]. Many authors like Gruber [11] identified three main categories of semantic heterogeneity in the context of data integration that can appear: confounding conflicts (e.g., concepts are defined in different), scaling conflicts (e.g., using different units for the same concept), and naming conflicts (e.g., synonyms).

Noy [19] identified three major dimensions of the terms in the application of ontologies for supporting semantic integration: the task of finding mappings (semi-)automatically, the declarative formal representations of data models, and reasoning using these mappings. There exist two main architectures for mapping discovery between ontologies: 1. It is possible to create a general upper ontology which is agreed upon by developers of different ontologies. 2. Examples for ontologies that are built specifically with the purpose of being formal top-level ontologies are the Suggested Upper Merged Ontology (SUMO) [18] and DOLCE [10].

There are approaches comprising heuristic-based or machine learning techniques that use various characteristics of ontologies (e.g., structure, concepts, instances) to find mappings. These approaches are similar to approaches for mapping XML schemata or other structured data [4, 6]. The declarative formal representation of mappings is facilitated by the higher expressive power of ontology languages which provide the opportunity to represent mappings themselves in more expressive terms.

Urbach and Gruninger [23] identified four major categories of ontologies for use in data interoperability and organization: 1. An ontology can be used to provide a shared and common understanding of a domain that can be communicated between people and application systems [8]. Given the vast number of non-interoperable tools and formats, a given company or organization can benefit greatly by developing their own domain ontology for authoring, and then developing translators from this ontology to the terminology required by the various target systems. While it is safe to assume there will not be global ontologies and formats agreed by all possible stakeholders, it is nevertheless possible to create an ontology to be used as a neutral interface for translating among various formats. There is a growing interest in the idea of "Ontology-Driven Software Engineering" in which an ontology of a given domain is created and used as a basis for specification and development of software [19]. The benefits of ontology-based specification are best seen if there is a formal link between the ontology and the software. To facilitate search, an ontology may be augmented with a repository of documents, e.g., data, and AI communities [6]. One of the most important and most actively studied problems in semantic integration is establishing mappings (also called alignments) between vocabularies of different data repositories.

III. INDUSTRY USE CASE

In cooperation with industry partners in the production automation domain we conducted the project "Simulator for Assembly Workshops" (SAW) [15], which simulates complex, continuously changing production automation systems by scheduling sequences of transport and machine tasks over 100 times faster than the actual hardware. The SAW simulator has been extended with real hardware components to ensure simulation validity for real-world production automation system. The main context comes from different backgrounds work shops and can be accessed through a better automated access to online data models which is currently only possible via workshops themselves as the data models are not well.

1. Illustrates sources of semantic gaps between: 1. A stakeholder domain layers with different local ontologies and design run-time views which are semantically well connected. The data model in our case an extended Engineering Knowledge Base (EKB) [17], which contains the common domain concepts to bridge the semantic gaps in the stakeholder terminologies and design run-time views.

2. Stakeholder layers in Figure 1 are a): the business production planning to fulfill customer orders by the operation layer (O) for the workshop tasks; b) the workshop layer for coordinating the complex system of transport and machines to assemble smaller basic products, and c) the knowledge base that models the data models, e.g., XML or OWL, which contain detailed information about the manipulators and their tasks. The stakeholders are divided into two parts based on the time those layers worked on, namely design time (development) and run time (usage).

UC-1. Translation between local stakeholder terminologies. The business manager on the business layer receives customer orders and schedules work tasks to the coordinator in the workshop layer. While they have a defined interface for exchanging work task information, they use local terminologies for concepts that are only occasionally needed to resolve scheduling issues, e.g., reference to specific customer orders if limited workshop capacity does not allow to fulfill all work tasks in a shift and negotiation on which tasks have higher priority are necessary to determine which workshop orders will be fulfilled. It is necessary to bridge different terminologies, translations are necessary to automate references to customer orders between stakeholders A business and workshop layers.

Figure 1: Sources of semantic gaps between stakeholders: domain layers, design-run time views; the data model contains common domain concepts to bridge semantic gaps.

2 Resource Description Framework: http://www.w3.org/ RD F/
3 Web Ontology Language: http://www.w3.org/2007/OWL

Figure 1 (right hand side) illustrates part of the data model that represents common domain concepts for the use cases in the SAW data diagram for assembly. The bottom box of each data element shows which stakeholder layer (B, W, and O) needs this data element to conduct their tasks and when: at Design Time (DT) or Run Time (RT). From the SAW project we derived the following use cases that illustrate semantic gaps between stakeholders and how to overcome these gaps using ontology-based approaches.

UC-2. Run-time measurement: 1. A data representation and analysis for design model improvements. If an engineering knowledge base is available to support run-time decisions with design knowledge, it is easy to also provide all kinds of run-time measurements linked to design elements, e.g., actual capacity of infrastructure, to iteratively improve the accuracy of design estimates with feedback from run time.
IV. RESEARCH ISSUES

The general idea of Ontology Areas (OAs) is to structure a comprehensive ontology into smaller building blocks with the following benefits for the designer and user of the ontology:

- A smaller ontology compared to the infrastructure.
- A subset of data on an OA that contains the minimal necessary knowledge for a specific task can be selected from a comprehensive ontology to facilitate more efficient use and storage.
- We expect a smaller ontology (consisting of selected OAs) to exhibit lower cognitive complexity for designers who work with ontologies to make tools that support the automation of stakeholder tasks.
- Specific OAs can contain more volatile ontology elements and thus make the design of the overall ontology more variable against changes.

As measurement criteria for evaluation we use the size of an ontology (and an OA) by counting the number of facts and relationships. In our study context the comprehensive ontology consists of a) the production automation domain concepts (i.e., data model in Fig. 1) for design-time and run-time elements; and b) stakeholder extensions to the data model, such as local terminologies and mappings, for all stakeholders.

We used the following guidelines to design the OAs:
- a) concepts that a particular stakeholder (in business, workshop, or operation layer) needs to fulfill its typical tasks in order to achieve objectivity and use the OAs; b) concepts that differ between concepts that may change in different project contexts; c) concepts that belong together in manageable chunks from more volatile run-time concepts; and d) structuralizing volatile run-time data by manageable time intervals depending on the frequency of data changes. According to these guidelines examples for concrete OAs are the design-time concepts of a business stakeholder and the run-time terminology of a work shop stakeholder.

Furthermore, we derive the following research issues (RIs) to investigate the benefits of an ontology structured with OAs compared to an ontology without OAs.

R1a: Comparison between local stakeholder terminologies. The ontology approach is allowing to use their local terminology to communicate with other stakeholders. For this task sufficient OAs need to contain for the communicating stakeholders the concepts that they use in their universe of discourse (see also in Fig. 1. the data elements and their link to associated stakeholders), local terminologies, mappings between local terminology elements and common domain concepts (on class level).

R1b: Evaluate the efficiency of the minimal ontology with OAs to the efficiency of the overall ontology in the study context to conduct the translation task.

The other use cases additionally benefit from making links between design-time and run-time data elements available at run-time.

UC-1. Translation between local stakeholder terminologies. The stakeholders of the production automation systems need to work together to achieve their goal. A common data schema is not possible because the stakeholders usually use different data formats, local terminologies and tools to access the data from the system. The ontology (EKD - Engineering Knowledge Base) plays a role as a common domain concept, where the local terminologies from the stakeholders will be mapped to. By mapping each local terminology to the ontology, the system can translate the local terminologies from one stakeholder to the other stakeholders. The translation would be the name of function, some names in the argument of the function, different data format, or the meaning of some parameters. However, the complexity of the ontology may increase when the number of the terminologies and the stakeholders is also increases, since the ontology should store all terminologies, the mappings and the common concepts.

By using the ontology area, the stakeholder can take a small part of the ontology that he really cares and solving his task with the same results but less complexity than by using the full ontology. The example is illustrated on figure 2.

Figure 2. Translation between Business Terminology and Workshop Terminology.

The business stakeholder has a local terminology “Client Order”, while the workshop stakeholder has a local terminology “Business Order”. Both have a common concept “Customer Order” in the Ontology Area. The local terminologies will be mapped to the class Customer Order in Section 4.

UC-2. Run-time measurement data representation and analysis for design model improvements. In the study context this task can be divided into two parts: the collection of run-time data points, e.g., on process characteristics and quality, and the infrastructure, helps to provide data for future design improvements, e.g., for more realistic planning and more efficient system configurations. The data processing and quality management personnel, who need to conduct the data analysis procedures, do not know in advance precisely which analysis functions they will need. Thus, a considerable amount of raw data would be beneficial to store in the ontology. Different design-time relationships and run-time data together. Unfortunately, even moderate data collection (600 data points) at reasonable frequency (e.g., one measurement every second) leads over a shift of 8 hours to a number of run-time data elements that easily exceeds the size of the design-time data elements in the ontology.

OAs that are designed to hold all measurement instances of a data element in a certain time interval (e.g., one minute) allow to keep the complexity of the ontology needed for analysis manageable. Only the OAs that contain relevant run-time measurements for a given analysis need to be considered.

R2a: Determine the minimal complexity of OAs to support a specific data analysis task more efficiently, such as calculating process characteristics. Compare the result with OAs to the (cognitive) complexity of using a whole ontology.

R2b: Compare the efficiency of the minimal ontology with OAs to the efficiency of the overall ontology in the study context to conduct the data analysis task.

V. ONTOLOGY AREAS FOR BRIDGING SEMANTIC GAPS

In this section we provide more detail how to address the use cases with an ontology that uses OAs as basis for the evaluation of the RIs in Section 6.

An ontology area is a subset of ontology as a building block that can solve a certain task interval (e.g., one minute) to allow to keep the complexity of the ontology needed for analysis manageable. Only the OAs that contain relevant run-time measurements for a given analysis need to be considered.

R1a: Compare the complexity (size) of the minimal ontology with OAs to the complexity of the overall ontology in the study context.

R1b: Compare the efficiency of the minimal ontology with OAs to the efficiency of the overall ontology in the study context to conduct the translation task.

The translation is just one example for transcoding in general. OAs for this use case would just consider the parts of the ontology for the stakeholders involved (see Figure 2): stakeholder extensions, local terminologies and mappings, which can more easily be added and removed from an ontology as stakeholders change in a particular context. The evaluation for this use case will be explained on section 6.

UC-2. Run-time measurement data representation and analysis for design model improvements. Run-time measurement information can be used to make design time information more accurate. Variational information like run-time measurement can produce large amounts of data which would make a single ontology unnecessary large and slow down the performance of the ontology. The need for storing a high volume of run-time measurement data in the ontology occurs if the trace of future domain analysis procedures are not known at the time of measurement.

Partioning of the ontology in areas of similar volatility allows building partial ontologies for the task or query at hand. Run-time measurement at the frequency of 1 data point per second provides 30,000 data points of shift of 8 hours. If this is too much information for the ontology to hold, it is possible to define OAs for smaller time windows, which allow including the data for a certain time frame to be loaded into the ontology for data analysis as needed without exceeding the capacity of the ontology.

Semantic gaps between run-time measurement and design-time information occur when we have data elements from the interest of one machine with run-time data from another machine understandable documentation for the design of the interface. To solve this problem, we first give meaning to run-time data that are not seen by the entities involved and then provide a link from run-time to design-time semantics.

For example, to find out the maximum process time of certain machine functions, we can measure the process duration of that machine function in one shift, and then collect sufficient and still manageable data. The measurement result is an event named “process” that consists of the id, the batch number, the date and timestamp of the measurement. Listing 1b shows several measurement results that can be obtained by filtering run time data. The real data themselves is a very long list.

Listing 1b. Run-time event data with semantic annotation.

```
<table>
<thead>
<tr>
<th>Job</th>
<th>Description</th>
<th>Start Time</th>
<th>End Time</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A1</td>
<td>10:00</td>
<td>10:30</td>
<td>30 min</td>
</tr>
<tr>
<td>B</td>
<td>B1</td>
<td>10:30</td>
<td>10:45</td>
<td>15 min</td>
</tr>
<tr>
<td>C</td>
<td>C1</td>
<td>10:45</td>
<td>11:15</td>
<td>30 min</td>
</tr>
</tbody>
</table>
```

To calculate the maximum process time of certain machine function, first, we should calculate each process time by using predicate “process_time” to find the difference between the timestamp of “stop” status and the related timestamp of “start” status from the same machine function and batch number, and then keep it in the list of process time “proc_time”. Then with using the predicate “maxprocess” we will find the
maximum value of process time of certain machine function (MFU) from the list of process time.

Listing 2b. Example analysis rule on run-time data.

```
max(X(0:T), Y(0:T)) > Y;  % Y
max(X(0:T), Y(0:T)) < Y;  % X
maxlist(X);  % X
maxlist(X);  % X
maxlist(X, Y);  % X, Y
maxlist(maxlist(X, Y));  % X, Y
maxlist(X(0:T), Y(0:T));  % X, Y
proces_list(MFU, S1, T, D);  % MFU, S1, T, D
T - 1 - X;  % T - 1
list_of_process_time(MFU, S1, T);  % MFU, S1, T
```

For `query`, for example we want to know the maximum process time of `MFU`. The result can be seen on Listing 2c.

Listing 2c. Result of data analysis.

```
mapprocess(MFU, T)
T = 0.1
```

The machine function entity in design time consists of the id and process time attributes. Usually the values of process time attributes come from estimation, but by using run-time measurement, we can compare the previous design-time estimates to actual run-time data for analysis on design improvements.

The example above is simple enough to conduct a statistical analysis at run time, but for more complex statistical analyses, a solution for storing large amounts of data in an ontology may be necessary, which would facilitate ontology size and decrease the ontology reasoning performance. OAs allow to manage stacks of run-time data elements and keep the size of ontology within well-performing capacity ranges.

V. EVALUATION AND DISCUSSION

We have implemented the OAs from the SAW ontology using Protege 3.3.1. The SAW ontology consists of 24 classes and 3,000 instances. Simulations of production automation system evaluation. The system will compare the measurement of the whole ontology and the ontology areas for three different use cases, namely 1, 5, and 50, as follows.

UC-1: Translation between local stakeholder terminologies. We compare the complexity (size) of the minimal ontology with the complexity of the overall ontology in the study context. For the minimal ontology with OAs, the business and workshop stakeholders have local terminologies of 300 and 400 words, respectively. Both need 100 words to communicate with each other. There are 200 to 700 data elements representing common knowledge, and 200 words for mapping from both local terminologies to the common concepts. Totally, 1,100 to 1,600 entities are needed for the OAs.

Meanwhile, the comprehensive ontology for 6 stakeholders consists of around 1,800 words for local terminologies and around 300 words to communicate with each other. There are 1,600 words of common knowledge, and 600 to 1,800 words for mapping of all local terminologies to common concepts. In total, the comprehensive ontology consists of 4,200 to 5,400 words. This demonstrates that the comprehensive ontology size is 20 to 30% of the comprehensive ontology.

We can compare the efficiency of the minimal ontology with OAs to the comprehensive ontology in conducting the translation task as follows. To process 100 words of translation results from 200 words of mapping, the OAs needs 3 operators of varying to those mapping.

The comprehensive ontology needs the run-time data (300 words) with 3 operators of query as well. But the query should be applied to mapping (600 to 1,800 words). With OAs we can reduce the size of mapping and make the operation faster.

UC-2: Run-time measurement and analysis for design improvement. For evaluation, we will determine the minimum of OAs to support a specific data analytics task more efficiently, such as calculating process characteristics. Then we will compare the result with OAs to the (cognitive) complexity using a comprehensive complexity. In the OAs of the specific task, for 1 volatile entity the run-time measurement consists of 30,000 data points per shift. In the overall ontology, there may be more entity, e.g., 300,000 data points in one shift. By using the OAs, the user can focus only on entity that he needs, and thus reduce the complexity of data handling considerably.

The efficiency of the minimal ontology with OAs is compared to the efficiency of the overall ontology in the case to conduct the data analytics task as follows. In the OAs, to obtain 5 data points per shift, it needs to run 3 operators of query over 30,000 data points at one shift. Hence 18,000 operations on data points are needed to obtain one of the measurements. In the overall ontology, to obtain 20 data points analysis, it needs to run 7 operators of query over 300,000 data points at one shift. Hence 45,000 operations on data points are needed to obtain one of the measurements. OAs is notably more efficient than that over the whole data.

Lesson learned. From the experiences with these use cases, we can learn the following lessons.

Building a small ontology for a task. As OAs allow focusing on the content of interest in a specific stakeholder task, the resulting ontology only contains the necessary knowledge for that particular task. A smaller ontology is often also more efficient to handle, making it easier to include in other projects. In the context of our work, this meant that the OAs could be used as a new workshop layout, new machines, or new between machines.

VI. ACKNOWLEDGMENT

We thank our colleagues at ACIN and TU Prague, for inspiring discussions, and the SAW for providing the application environment space.

REFERENCES


