SEMI-AUTOMATED SOFTWARE INTEGRATION
An approach based on logical inference

Mikhail Kazakov
Research Division, Open CASCADE S.A., 4, rue Rene Razel, 91400 Saclay, France

Habib Abdulrab
PSI laboratory, INSA de Rouen, BP8, place Emile Blondel, 76131 Mont Saint Aignan, France

Keywords: Enterprise integration, formal specification, logical inference, description logics

Abstract: The paper addresses a problem of semi-automated enterprise application integration. More close we discuss a problem of integration of numerical simulation components in the area of manufacturing engineering information systems. We propose an approach that is based on annotation of software interfaces with formal logical specifications. Logical inference procedure is used to choose appropriate enterprise software component depending on client requests. First, we discuss the problem and difficulties of integration of numerical simulation solvers and manufacturing engineering solutions in general. This is followed by description of the methodology of semi-automated integration based on use of description logics.

1 INTRODUCTION

Nowadays, software integration is one of the most important and complex areas of software engineering. The complexity of integration problem can be explicitly seen in the domain of enterprise information systems where many legacy software exist, new systems occur, existing systems shall be integrated together. Recently, the domain of integration of manufacturing engineering components raised new challenges:

- Complexity of software algorithms leads to presence of legacy software.
- Most of the times the exact specification and behavioral model of engineering software are known only by domain experts and are hidden from developers who have only the external description of API (or a protocol) of the systems. This leads to semantic mistakes during creation of connectors among software entities, consequently increasing the time and cost of integration process.
- Specifications of APIs and protocols come often in a form where many specific abbreviations are used and interfaces are not designed according to the best practices of software engineering.

Historically, manufacturing engineering domain is very inertial. It accepts new software technologies with significant impedance. However, the market forces manufacturing engineering area to adopt common enterprise application integration (EAI) practices such as message-oriented middleware, shared application servers, distributed transactions with formal data verification and many others.

Following this demand, the EAI area needs to research and adopt new techniques that will help to cope with above-mentioned challenges and reduce integration time and cost. This is especially important now, when enterprise integration market is moving towards Web Services technology.

Analysis of the above-mentioned challenges of enterprise integration of manufacturing systems shows that one of the biggest problems is presence of a big amount of possible components and a big amount of domain terms and semantics that are known to domain experts but:

- are not always known by architects/developers
- have multiple different data representations in different systems
- can look very similar but mean different thing for domain experts.

Kazakov M. and Abdulrab H. (2004).
SEMI-AUTOMATED SOFTWARE INTEGRATION - An approach based on logical inference.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 527-530
DOI: 10.5220/0002638005270530

Integration of numerical simulation solvers poses even more difficulties:

- Specifications of solvers are understandable only by someone with background in thermal physics.
- Same terms of thermal analysis can mean different things among different solvers (example: Maximum of temperature).
- Data needed by one of the solvers is missing
- Each solver has many different parameters. Solvers have interfaces implemented with different technologies.

We can continue this list of difficulties; however, it is clear that automated support in partial mapping among enterprise components would be appreciated.

This paper sketches a new methodology of semi-automated integration of engineering components. The methodology is based on formal specification of software components using logical languages. Inference procedure obtains a partial mapping among software components with consequent generation of connector code. The mapping is proposed to a person (integrator) that shall validate and complete the integration process. We describe a software prototype that implements the methodology.

2 SEMI-AUTOMATED INTEGRATION

We argue that separation of integration information onto three following layers is useful:

- Domain knowledge – specification of real-world semantics. Example: A thermal solver has to accept heat conductivity of material as one of its parameters. This information is almost not present in the information systems and this is one of the biggest obstacles for automation of integration process.
- Specification of interfaces – semantics of mean of access to a specific component. Example: "ISolver" is an interface with "Calculate" method.
- Technological information – technology-specific data that conforms to specification of a technology and defines the way of physical interoperation with the technology.

This separation seems to be evident; however, the division is extremely important for semi-automated approach. Specificity of information that is used on one of the levels is usually not needed on others (EJB technology itself does not need to care about details of physics and physics does not need to know its implementation details). Thus, we can separate means that we use to work with information of each level. Adding the domain knowledge as an equal knowledge component is extremely important for semantic-aware integration.

In order to specify all the above-mentioned information types we proposed to use formal logical specifications. The use of logics for specification of computation-independent models is very reasonable due to the high level of abstractions of such models and their direction towards description of structure rather than behavior. Hereinafter we use the term "ontology" to refer to a formal logical model.

SHIQ description logic (Baader, 2003) is used as formal logical representation mechanism. SHIQ logic has proven to be decidable, has at least two working implementations and is widely accepted by the Semantic Web community. DAML+OIL and its successor – OWL are the XML-based formats for persistent representation of ontologies.

Ontological specifications are separated onto several layers according information layers:

- Domain ontology – a logical specification of application domain.
- Interface ontology – logical specification of application programming interfaces (APIs).
- Technological ontology – this layer abstracts software components from operating system and mediates the components over distributed environment.

In addition, we specify a mediation ontology – a specification of how reasoning, querying and binding among ontologies is done within our methodology.

The domain ontologies are created by domain experts. It is supposed, that domain ontologies are not just the terminological taxonomies (hierarchical listing of terms), but are as complete as possible logical model. That assure that new knowledge can be inferred by a logical reasoner and integration can be performed even in non-evident cases.

Interface ontologies consist of three parts. The first part (IO1) is constant and specifies common notions of software interfaces (APIs), such as method, call, parameter, argument, return value, interface, etc. This part is very close semantically to the UML representation of the same information.

The second part of interface ontology (IO2) is generated automatically from definitions of APIs of concrete components that need to be integrated. It follows semantics of APIs (expressed in some language, for example IDL, Java, etc.). APIs are expressed in terms of the first part of interface ontology.

The third part of the interface ontology (IO3) represents a binding among interface and domain

ontologies. The binding is manually created by domain experts and software engineers. On this step, a connection among interfaces and their real-world semantics is done. For example, the following axioms can be defined: "Thermal_solver" concept from IO2 is subsumed by a "thermal solver" concept from domain ontology; the "Thermal_solver" concept is in relation of "represented by" with "thermal solver"; etc. These propositions define how specific concepts and relations from interface ontologies are related to specific concepts and relations from domain ontology. As one can guess, the success of reasoning depends a lot on completeness of this binding.

Specification of technological ontologies is possible without using of logical languages. Research results in auto-configuration environments that can be reused instead. In the current prototype, we avoid using technological ontologies due to high complexity of ontological models. Configuration parameters are pre-defined for research purposes.

Mediation ontologies consist of three parts. The first part (MO1) is static and contains semantics that are needed for successful reasoning and integration. The main goal of this ontology is to specify how a "client query" can bound to "service answer" and to specify how to achieve this binding via inference on interface, domain and technological ontologies. In addition, this part contains axioms needed to assure the structural validity of static parts of other ontologies (first parts of interface and technological ontologies) by constraining them and introducing the checks of axioms that shall never be false in a correct model, but always be false in incorrect models.

The second part of the mediation ontology (MO2) plays the same role as a third part of interface ontology but is specific to clients (a part that performs calls to services) and shall be written manually while specifying client request to call a service. A domain expert shall connect the concepts and relations of domain ontology with concepts and relations of the first part of the mediation ontology and to the first part of interface ontology.

The third part of mediation ontology (MO3) is generated automatically and represents information from other ontologies in terms of the MO1. For example, it generates axioms that will be used to infer the possibility of binding of a client to a service for each pair of client and server methods. Details on this part are not present in this paper due to problems with size.

A set of four ontologies shall be created for each component of enterprise system to be integrated (depending on its role to be client or service, a second part of mediation ontology may be used or not).

We propose the following scenario of semi-automated integration using our methodology:

- Ontology editor prototype is used to create all the manual parts of ontologies and connect them with static parts
- Integration prototype (hereinafter mentioned as "integrator") is used to annotate these manual parts to specific software interfaces
- The integrator is used to generate the automated parts of interface and technological ontologies from the interfaces
- Then the integrator takes all the manual and static parts of above-mentioned ontologies for each component to be integrated and merge them together. On this step we receive an common ontology that represents the specification of a set of enterprise components to be integrated
- Addition of missing parts of interface and technological ontologies is performed in order to have a binding among ontologies
- The structural and logical verification of this set is performed by a theorem prover
- Then a third part of mediation ontology is generated in order to create binding among client and server roles within the system
- Reasoning is performed to find possible mapping among client and server parts
- The result of such reasoning is analysed in order to receive names of interfaces, methods, and configuration parameters to be integrated
- This information is converted back to integration software that generates connectors among different systems.

Communication among clients and servers can be performed in two principal modes: context-free (or context-less) and context-aware (context-full). In the context-free mode, no client-specific information is kept between two consequent communication sessions among clients and server. In the context-free mode, even if the state of server exists and changes, this change is not important and does not influence the future client-server communication. In context-full mode, the client server communication may depend on change of state of the server and/or presence of the context of the communication session.

Our logic-based semi-automated integration methodology is intended to integrate context-free software components. In this case, we presume that client requests to components will not depend on components themselves. It is important, since semi-automated client-to-service binding mechanism may easily return two different servers for two

consequent client calls. It is necessary to be sure, that the second call will not depend on the first one.

The requirement for having context-less enterprise components is well suited for Web Services paradigm since this technology is service based (request – response model).

The software support of the methodology consists of two prototypes: DL-workbench and DL-Integrator.

DL-workbench is a meta-model based platform for definition and edition of data structures that has default user interface for ontological manipulation. On the bottom level of the platform, one can find a metadata description language (meta-model) with meta-data repository. The meta-model can define structures of ontological language, interfaces representations, data mappings and other structural formalisms. Data that is managed by the platform is driven and constrained by structural models that are defined via meta-modeling language.

The DL-workbench is an open source software product that can be downloaded from the http://www.opencascade.org/dl-workbench web site (DL-workbench, 2003).

DL-integrator is a software prototype that implements the integration process of the proposed methodology. The DL-integrator is based on the DL-workbench architecture. DL-integrator specifies supplementary models for representing the WSDL interfaces and several other data formats that are specific to textual interfaces of numerical simulation solvers. The use of the same meta-model repository allows easy and manageable implementation of associations among ontologies and interface information. Details on the methodology and software prototypes can be found in (Kazakov, 2002) and (Kazakov, 2003).

3 RELATED WORKS AND CONCLUSION

Several research projects exist in the area of semi-automated or automated integration. NIST MEL laboratory (Ray, 1999) project has been conducting since 1999 and has 10 years goal of feasibility proof of use of automated methods in manufacturing engineering enterprise networks. This project comes also to the conclusion that semi-automated approaches are feasible. NIST focuses mostly on use of Express language. By the year 2003, NIST did not start working on problem of integration of numerical simulation solutions yet.

The semi-automated composition approach for Web Services is described in (Sirin, 2003). The authors present a mechanism of reasoning on top of specifications of Web Services using DAML-S language. While having the impression of similarity, our approach is very different. The authors do not separate ontologies on several layers and do not provide any generic mechanism to bind the code. Separation of ontologies on interface and domain parts gives us high level of control over the complexity of numerical simulation solutions. DAML+S orientation provides service-based view on components that does not fit our goal of integration of interfaces where more complex scenarios shall be implemented.

In this paper, we have presented our methodology of semi-automated integration of stateless software components. By creating of this methodology, we have proven feasibility of use of the methodology by prototyping. The approach can be applied to integration of context-free components.

Following our methodology, the domain experts can specify scientific and engineering components with domain semantics on a high level. Software architects can reuse these specifications to integrate software components. The methodology opens up perspectives for dynamic composition of software components.

Authors hope that after some time, the semi-automated and automated enterprise integration techniques will be widely used by EAI domain.

REFERENCES

Baader F, et all. 2003. *The description logics handbook: Theory, implementation and applications*, Cambridge Unviersity Press, ISBN: 0521781760.

DL-workbench., 2003. *Project web site*. Online: http://www.opencascade.org/dl-workbench

Kazakov M., Abdulrab H., Babkin E., 2002, *Intelligent integration of distributed components: Ontology Fusion approach*, In proceedings of CIMCA 2003 conference, pp 611-622, ISBN 1-740-88069-2

Kazakov, M., Abdulrab H., 2002, *On semantic-enhanced middleware,* INSA de Rouen, Internal report [KAZ,02e]

Kazakov M., Abdulrab H., 2003, *A meta-modeling approach to ontological engineering: DL-workbench platform*, In proceedings to MIS 2003 conference, Springer-Verlag

Ray S., 1999. *The Future of Software Integration: Self-integrating Systems*, NIST MEL report

Sirin E., Hendler J., Parsia B., 2003, *Semi-automatic Composition of Web Services using Semantic Descriptions*. In proceedings of "Web Services: Modeling, Architecture and Infrastructure" workshop in conjunction with ICEIS2003