Chapter 2
The Ontological Stance for a Manufacturing Scenario

Michael Gruninger
University of Toronto, Canada

EXECUTIVE SUMMARY

The semantic integration of software systems can be supported through a shared understanding of the terminology in their respective ontologies. In practice, however, the author is faced with the additional challenge that few applications have an explicitly axiomatized ontology. To address this challenge, we adopt the Ontological Stance, in which we can model a software application as if it were an inference system with an axiomatized ontology, and use this ontology to predict the set of sentences that the inference system determines to be entailed or satisfiable. This chapter gives an overview of a deployment of the Process Specification Language (PSL) Ontology as the interchange ontology for the semantic integration of three manufacturing software applications currently being used in industry—a process modeller, a process planner, and a scheduler.

MOTIVATION

The necessity for software applications to interoperate has become crucial to the conduct of business and operations in organizations. In practice, however, interoperability is difficult to achieve, since each of these applications utilizes information in a different way, and the knowledge representation formalisms inherent to these applications differ. In particular, existing approaches to interoperability lack an adequate specification of the semantics of the terminology used by the software applications, which leads to inconsistent interpretations and uses of knowledge.

The development of ontologies has been proposed as a key technology to support semantic integration. Ontologies are logical theories that provide a set of terms together with a computer-interpretable specification of the meanings of the terms in some formal logical language. In this way, we can support the semantic integration of software systems through a shared understanding
of the terminology in their respective ontologies, in the sense that a semantics-preserving exchange of information between ontologies requires mappings between logically equivalent concepts in each ontology. The challenge of semantic integration is therefore equivalent to the problem of generating such mappings, determining that they are correct, and providing a vehicle for executing the mappings, thus translating terms from one ontology into another. The emphasis in this article will be on the verification of the correctness and completeness of semantic mappings rather than on the specification of the mappings (as in (Giunchiglia et al., 2007), (Kalfoglou & Schorlemmer, 2003)).

In practice, we are faced with the additional challenge that few applications have an explicitly axiomatized ontology. To address this challenge, we adopt the Ontological Stance, in which we can model a software application as if it were an inference system with an axiomatized ontology, and use this ontology to predict the set of sentences that the inference system determines to be entailed or satisfiable.

In the first part of this article, we discuss a set of sufficient conditions that a neutral interchange ontology must satisfy in order to support this approach to the generation and validation of semantic mappings. In the second part of the article we give an overview of a deployment of the Process Specification Language (PSL) Ontology as the interchange ontology for the semantic integration of three manufacturing software applications currently being used in industry—a process modeller, a process planner, and a scheduler. The semantic mappings between the applications’ ontologies and PSL are semi-automatically generated from invariants (properties of models preserved by isomorphism) and verified prior to integration. The correctness of the application ontologies are validated using the notion of the Ontological Stance.

### FORMALIZATION OF SEMANTIC INTEGRATION

When software applications communicate with each other, there needs to be some way to ensure that the meaning of what one application accepts as input and output is accurately conveyed to the other application. Since the applications may not use the same terms to mean the same concepts, we need a way for an application to discover what another application means when it communicates. In order for this to happen, every application needs to publicly declare exactly what terms it is using and what these terms mean; this specification is commonly referred to as the application’s ontology. Moreover, this specification must be accessible to other software applications. Thus, we require that the meaning be encoded in a formal language, which enables a given application to use automated reasoning to accurately determine the meaning of other applications’ terms. For example, if application 1 sends a message to application 2, then along with this message is a pointer to application 1’s ontology. Application 2 can look in application 1’s ontology to see what the terms mean, the message is successfully communicated and the application’s task is successfully performed. Complete understanding will occur between the applications only if they share the semantics of all the terminology used in the content of the messages that they exchange or the information sources that they access.

We can therefore say that two software applications will be interoperable if they share the semantics of the terminology in their corresponding ontologies. Sharing semantics between applications is equivalent to sharing models of their ontologies; two ontologies are equivalent if they have isomorphic sets of models. Nevertheless, applications do not explicitly share the models of their theories; instead, they exchange sentences in such a way that the semantics of the terminology of these sentences is preserved.