2005 3rd IEEE International Conference on Industrial Informatics (INDIN)

Towards 'Ontology'-based Software Engineering for Multi-site Software Development

Wongthongtham, P.[1], Chang, E.[2] and Dillon, T.S.[3]

[1,2]School of Information Systems, Curtin University of Technology, Australia, e-mail : (Pornpit.Wongthongtham, Elizabeth.Chang)@cbs.curtin.edu.au
[3] Faculty of Information Technology, University of Technology Sydney, Australia, e-mail: tharam@it.uts.edu.au

Abstract—**The purpose of this paper is to discuss the research issues related to multi-site distributed software development environments. We describe a potential approach in which we utilize 'Ontologies' as part of a communication framework for multi-site distributed software development environments. We organise software engineering concepts, ideas and knowledge, software development methodologies, tools and techniques into an - 'ontology' -, and use it as the basis for classifying the concepts in communication thereby enabling questions, problem solving and sharing solution development and knowledge to be shared between multi-site teams.**

Index Terms— **Software Engineering 'ontology', 'ontology' Development.**

I. Introduction

The widely accepted model of the centralized and single-site software development approach has been widely used by large and medium software development teams for many years. Traditional software development models cover environments where all necessary software development documents and source codes reside on a local server available to the developer over a Local Area Network (LAN). However, in today's global economy, collaborative software development, spanning multiple teams in multiple development locations, is the norm rather than the exception [1]. We often see that teams of developers working in today's software development span many cities, regions and sometimes across several continents. At present it is common for even small to medium sized software development projects to consist of two or more clusters of developers working across several distributed locations [2]. Chang et al. [3] state that existing systems development involves dynamic business modeling, utilize a variety of technologies, use a range of analysis and design methodologies, rely on complex project management skills and require increasing domain knowledge. In most single sites it is difficult to have competent people with skills in all these areas of software development. To successfully deliver a large complex computer based information system and to reduce the cost of software development, the practice is often to outsource the development to acquire the necessary skills. This means that specialized groups will have to work together from remote sites to achieve common integrated development goals[3-7].

'Ontologies' have become a popular research topic because of what they promise: a shared and common understanding of a domain that can be communicated between people and application systems e.g. software agents [7]. As such, an 'ontology' based software development methodology serves as a structure for an underlying knowledge base, allowing for interactions, comprehension and customization between multi-site teams.

In this paper, we look at multi-site distributed software development through 'ontology-based' software engineering. The paper is structured as follows: First, we discuss traditional approaches. We offer a brief definition on 'ontology'. We then explain our approach, 'ontology-based 'multi-site distributed software development in sections 4 and 5. The last section provides a conclusion and future work.

II. Traditional Approach

Traditional software process models such as the spiral model, V model or waterfall model; and traditional methodologies such as structured methods, object and component based software engineering [8], XP (eXtreme Programming) [9]; and prototype methodologies [8], as well as agent-based approaches [10] have not addressed the needs of multi-site, multi-team situation and its environment [11]. Most examples of software process models that are in existence assume a centralized approach to software development. This assumption is also present in many of the current software engineering methodologies which do not address multi-site distributed software development. Most project management approaches also do not consider multi-site distributed software development. We also observe that many technologies are not mature enough to facilitate multi-site distributed software development. The process models, methodologies, technologies and approaches cannot be followed in a linear way as suggested for multi-site distributed software development. In the last few years, we have been extensively involved in multi-site distributed software development and have identified these deficiencies with existing methodologies. We have tried to overcome the difficulties that frequently result in increased costs through the need for excessive travel to initiate face-to-face contact and in re-development arising from misunderstandings between the remote teams. In this paper, we will discuss a move away from the standard centralised assumptions and one that allows for different parts of the software development team to be physically at different sites but working on a common project. In doing so, we are propose a significant innovation in this area.

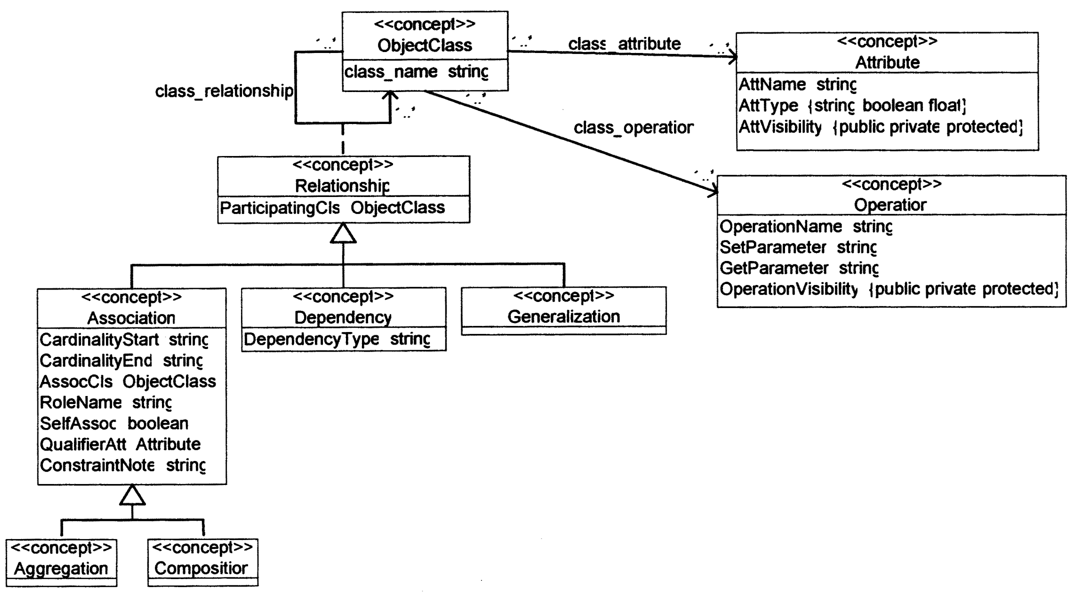

Figure 1: 'Object class ontology' model

III. ONTOLOGY DEFINITION

An 'ontology' is defined here as an explicit way of specifying some domain of understanding [12]. In other words, 'ontology' is a formal and declarative way of describing and representing concepts and their relationships within a certain subject area. An 'ontology' defines concepts by describing their properties and constraints on properties. In 'ontology-based' software engineering we propose two sub-ontologies (a) a 'Generic ontology' and (b) a 'Specific ontology'. The 'Generic ontology' is a set of software engineering terms including the vocabulary, the semantic interconnections, and many simple rules of inference and logic for software development. The Generic 'ontology' provides the vocabulary or names for referring to the terms used in software engineering. The purpose is mainly for knowledge sharing. The 'Specific ontology' is a specification of particular software development project with specific concepts and properties that are suitable to that particular project. The 'Specific ontology' provides project agreement that can be communicated between teams distributed across sites.

'Ontologies' simply serve to standardize and provide interpretations for machine-understandable content. Software engineering is a well-known discipline that encompasses software development and it should be able to record all definition in machine-interpretable manner.

IV. DEVELOPMENT OF 'ONTOLOGY-BASED' SOFTWARE ENGINEERING

In order to determine what is of interest in a domain and how information about it can be structured, firstly we define concepts or 'ontology' classes in the domain. The next step is to arrange the concepts in a hierarchy or to be precise in a subclass-superclass hierarchy and relationships among the concepts. Then we define properties or slots of the classes and the constraints on their values. The last step is to define individuals or instances and fill in slot values. To represent the 'ontology' for machine processing and exchange of information among machines or agents, we are use the 'web ontology language' (OWL). OWL has capability for capturing the semantic richness of a defined 'ontology' [13]. We use UML to model the ontologies and the communication architecture. The semantic of 'ontology' modeling help generalize the common semantic thus hiding implementation details. The main aim of the use of UML notations is simply to create a graphical representation of 'ontologies' to make it easier for others to understand. Note that this use of UML notations to model the underlying 'ontology' should be distinguished from its use in software development to model the application domain.

The 'ontology' that is defined can cover the subject content of software engineering discipline which can be divided into five sub areas i.e. software process, requirements engineering, design, development, and verification and validation. In this paper we will not include the areas of project management and business domains which form our future work. For this example we developed an 'ontology-based' software engineering model which will promote a consistent understanding of software engineering across a team of multi-site software developers spread around the world and also serves as a common curriculum. As you may notice, software engineering knowledge already exits therefore we have simply re-structured it as an - 'ontology' – providing a common basis for communication questions problem solving solution development and exchange of information between the multi-site teams.

We not only define the terms but also the properties of these terms are. For example, the term of object class in UML can define attributes and operations of classes and relationship among classes. Therefore, properties of the term 'object class' can be class_name, class_attribute, class_operation, and class_relationship. Figure 1 illustrates 'object class' 'ontology' model, a meta-model of 'object

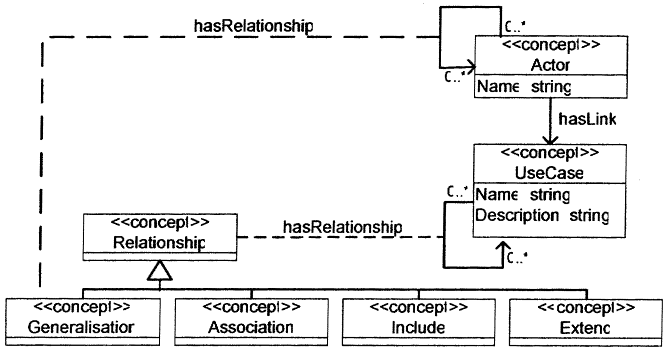

Figure 2: 'Use case ontology' model

class' model in UML. As you can see from Figure 1, the class_name property is Datatype property whose range can only be datatype value. Object property is distinguished from datatype property by whose range can be only instance. From the meta-model in Figure 1, object property is class_attribute, class_operation, and class_relationship.

A subclass-superclass hierarchy is A taxonomic relationship between a more general class and a more specific class. The more specific class is fully consistent with the more general element and contains additional information. For example, the term 'class relationship' in UML can be more specified by association, dependency, or generalization relationship. Association relationship can be better specified by aggregation or composition. Figure 1 shows the concept hierarchy of class relationship. Figure 2 depicts the 'use case ontology' model, a meta-model of the 'use case' model in UML. It shows relationships among actors and use cases within a system. 'Use case ontology' would be equally helpful as an alternative to the 'object class ontology'. Additionally there are five more ontologies that can be captured together with the object class diagram and 'use case' i.e. activity, state chart, package, sequence, and collaborative 'ontology'. These concepts are worth capturing in machine-interpretable manner.

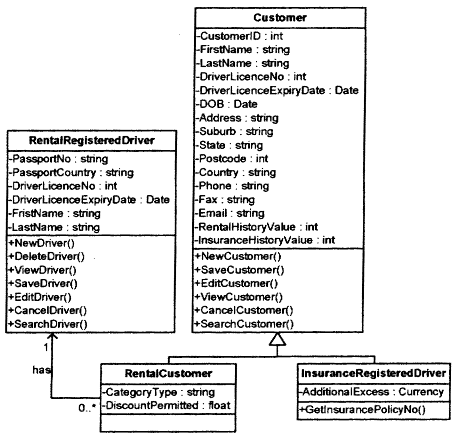

Figure 3: An example of class diagram put into the 'ontology' as instances

The last step of the development would be to add individuals as necessary. We have made an 'ontology' of software engineering and we now have a clear context to proceed. What we now need to do is to create data instances to be usable. Figure 3 shows an example of a class diagram for a particular project which we have used to map as instances of the 'object class ontology' model.

I am struggling to understand why we need it.

I think the system will be simpler for people to understand if we deleted the insurance registered driver.

My reasons for this are that the insurance registered driver is a sub type of the customer. This means that for every insurance registered driver object there must be a corresponding customer object. However, in the customer object we store values like customer type, insurance history value and rental history value. It does not make sense to have these values for the insurance registered driver. I also think people will be confused because we have the rental registered driver as an association with the rental customer (which is a sub type of the customer) but the insurance registered driver is a sub type of the customer.

Figure 4: An example of plain text communication.

V. 'ONTOLOGY-BASED' SOFTWARE ENGINEERING FOR MULTI-SITE SOFTWARE DEVELOPMENT

In this section we illustrate how 'ontology-based' software engineering facilitate multi-site distributed software development environment. Because there is no face to face communication, in multi-site distributed software development environments it is very difficult to develop a consistent understanding of the meaning of terminology or understanding of project agreement. Figure 4 shows a typical example of global communication which does not create consistent understanding of that particular project design. By using 'object class' 'ontology' of Figure 1 and instances of Figure 3 we are able to convert the plain text into a UML-like form that provides consistent understanding among multi-site developers. Software agents for example can be utilized to extract information from the 'ontology' described in OWL. To do so, software agent consults the 'object class ontology'. The 'ontology' shows how class is formed in the class diagram; each class contains a name, attributes, and operations; and relationships between the classes. Therefore, the software agent dynamically act to retrieve involved class name, involved class attributes, involved class operations, and involved relationships to draw a class diagram.

The 'ontology-based' software engineering is made explicit therefore becomes formalized in one way or another. Explanation of software engineering discipline is explicit (e.g. book form) but informal in knowledge. By using an 'ontology' we help to clarify the software engineering concepts thereby making them not only explicit but also formalizable and consistent for use by multi-site developers who would not have the opportunity to interact with each other to clarify meaning. The second advantage of the 'ontology' form is that it can be understood by machine-readable communication software.

VI. CONCLUSIONS AND FUTURE WORK

We have examined how 'ontology' can facilitate consistent understanding between multi-site developers. From the 'ontology' software repository, we can share components of software across multi-site distributed software development teams. Co-operation refers to the sharing of information and knowledge of the software production processes and how to support each other in the development process.

In future work we plan to try to expand our software development repository to help identify other components of collaboration such as communication, coordination, awareness, and interoperability by utilizing software agents.

VII. REFERENCES

[1] Capasso, R., Keeping Geographically Distributed Development Team in Sync. 2000, Rational Software.

[2] Audet, J. and G. Amboise, The Multi-Site Study: An Innovative Research Methodology. An online journal dedicated to qualitative research and critical inquiry, 2001. 6(2).

[3] Chang, Jayaratna, and Aisbett. Generational Change Model in the Software Life Cycle. in Proceeding of IASTED Conference on Software Engineering. 2003. Innsbruck, Austria.

[4] Bergland, G.D. A guided Tour of Program Design Methodologies. in IEEE Computer. 1981.

[5] Chang, Davis, and Chalup. A NEW LOOK AT THE ENTERPRISE INFORMATION SYSTEM LIFE CYCLE-Introducing the concept of generational change. in Proceeding of 5th International Conference on Enterprise Information Systems (ICEIS) 2003. 2003. Angers, France.

[6] Stuart, W. Evolutionary Project Management. in IEEE Computer. 1999.

[7] Davies, J., D. Fensel, and F.v. Harmelen, Towards the semantic web: 'ontology'-driven knowledge management. 2003: John Wiley & Sons.

[8] Sommerville, I., Software Engineering. 7th ed. 2004: Pearson Education Limited.

[9] Beck, K., Extreme Programming Explained. 2000: Addison Wesley.

[10] Wooldridge, M., Introduction to MultiAgent Systems. 1st ed. 2002: John Wiley & Sons.

[11] Chang, E. and T. Dillon. A Generation Model for Collaborated Distributed Software Development Paradigm. in To appear in Computer System Science and Engineering. 2003.

[12] Gruber, T.R. A translation approach to portable ontologies. in Knowledge Acquisition. 1993.

[13] McGuinness, D.L. and F.V. Harmelen, OWL Web 'ontology' Language Overview. 2004.

Towards 'Ontology'-based Software Engineering for Multi-site Software Development

Wongthongtham, P.[1], Chang, E.[2] and Dillon, T.S.[3]

[1,2]School of Information Systems, Curtin University of Technology, Australia,
e-mail : (Pornpit.Wongthongtham, Elizabeth.Chang)@cbs.curtin.edu.au
[3] Faculty of Information Technology, University of Technology Sydney, Australia, e-mail: tharam@it.uts.edu.au

Abstract—**The purpose of this paper is to discuss the research issues related to multi-site distributed software development environments. We describe a potential approach in which we utilize 'Ontologies' as part of a communication framework for multi-site distributed software development environments. We organise software engineering concepts, ideas and knowledge, software development methodologies, tools and techniques into an - 'ontology' -, and use it as the basis for classifying the concepts in communication thereby enabling questions, problem solving and sharing solution development and knowledge to be shared between multi-site teams.**

Index Terms— **Software Engineering 'ontology', 'ontology' Development.**

I. Introduction

The widely accepted model of the centralized and single-site software development approach has been widely used by large and medium software development teams for many years. Traditional software development models cover environments where all necessary software development documents and source codes reside on a local server available to the developer over a Local Area Network (LAN). However, in today's global economy, collaborative software development, spanning multiple teams in multiple development locations, is the norm rather than the exception [1]. We often see that teams of developers working in today's software development span many cities, regions and sometimes across several continents. At present it is common for even small to medium sized software development projects to consist of two or more clusters of developers working across several distributed locations [2]. Chang et al. [3] state that existing systems development involves dynamic business modeling, utilize a variety of technologies, use a range of analysis and design methodologies, rely on complex project management skills and require increasing domain knowledge. In most single sites it is difficult to have competent people with skills in all these areas of software development. To successfully deliver a large complex computer based information system and to reduce the cost of software development, the practice is often to outsource the development to acquire the necessary skills. This means that specialized groups will have to work together from remote sites to achieve common integrated development goals[3-7].

'Ontologies' have become a popular research topic because of what they promise: a shared and common understanding of a domain that can be communicated between people and application systems e.g. software agents [7]. As such, an 'ontology' based software development methodology serves as a structure for an underlying knowledge base, allowing for interactions, comprehension and customization between multi-site teams.

In this paper, we look at multi-site distributed software development through 'ontology-based' software engineering. The paper is structured as follows: First, we discuss traditional approaches. We offer a brief definition on 'ontology'. We then explain our approach, 'ontology-based 'multi-site distributed software development in sections 4 and 5. The last section provides a conclusion and future work.

II. Traditional Approach

Traditional software process models such as the spiral model, V model or waterfall model; and traditional methodologies such as structured methods, object and component based software engineering [8], XP (eXtreme Programming) [9]; and prototype methodologies [8], as well as agent-based approaches [10] have not addressed the needs of multi-site, multi-team situation and its environment [11]. Most examples of software process models that are in existence assume a centralized approach to software development. This assumption is also present in many of the current software engineering methodologies which do not address multi-site distributed software development. Most project management approaches also do not consider multi-site distributed software development. We also observe that many technologies are not mature enough to facilitate multi-site distributed software development. The process models, methodologies, technologies and approaches cannot be followed in a linear way as suggested for multi-site distributed software development. In the last few years, we have been extensively involved in multi-site distributed software development and have identified these deficiencies with existing methodologies. We have tried to overcome the difficulties that frequently result in increased costs through the need for excessive travel to initiate face-to-face contact and in re-development arising from misunderstandings between the remote teams. In this paper, we will discuss a move away from the standard centralised assumptions and one that allows for different parts of the software development team to be physically at different sites but working on a common project. In doing so, we are propose a significant innovation in this area.

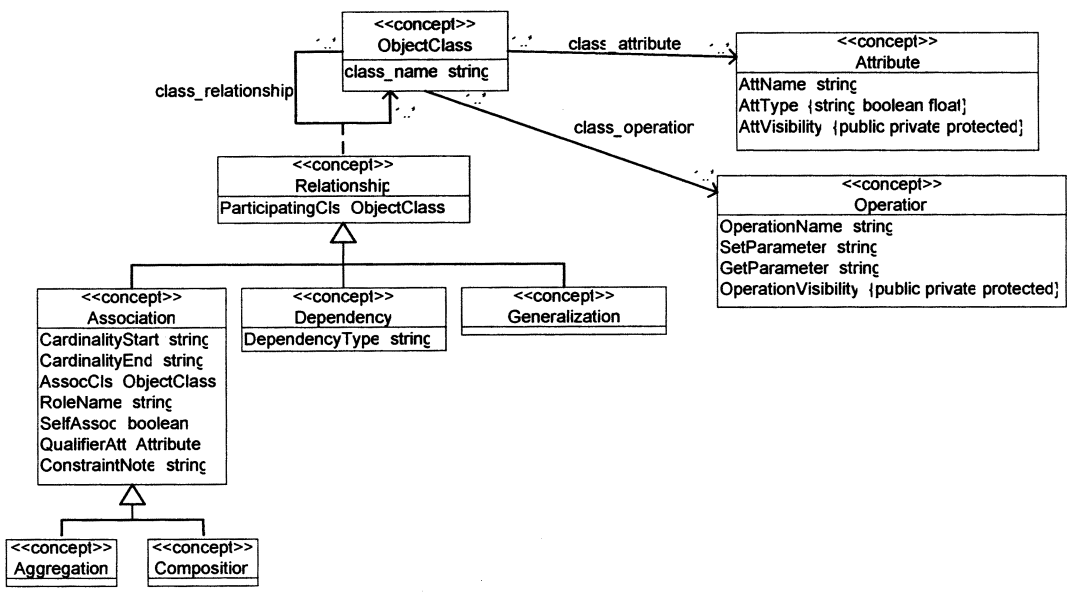

Figure 1: 'Object class ontology' model

III. ONTOLOGY DEFINITION

An 'ontology' is defined here as an explicit way of specifying some domain of understanding [12]. In other words, 'ontology' is a formal and declarative way of describing and representing concepts and their relationships within a certain subject area. An 'ontology' defines concepts by describing their properties and constraints on properties. In 'ontology-based' software engineering we propose two sub-ontologies (a) a 'Generic ontology' and (b) a 'Specific ontology'. The 'Generic ontology' is a set of software engineering terms including the vocabulary, the semantic interconnections, and many simple rules of inference and logic for software development. The Generic 'ontology' provides the vocabulary or names for referring to the terms used in software engineering. The purpose is mainly for knowledge sharing. The 'Specific ontology' is a specification of particular software development project with specific concepts and properties that are suitable to that particular project. The 'Specific ontology' provides project agreement that can be communicated between teams distributed across sites.

'Ontologies' simply serve to standardize and provide interpretations for machine-understandable content. Software engineering is a well-known discipline that encompasses software development and it should be able to record all definition in machine-interpretable manner.

IV. DEVELOPMENT OF 'ONTOLOGY-BASED' SOFTWARE ENGINEERING

In order to determine what is of interest in a domain and how information about it can be structured, firstly we define concepts or 'ontology' classes in the domain. The next step is to arrange the concepts in a hierarchy or to be precise in a subclass-superclass hierarchy and relationships among the concepts. Then we define properties or slots of the classes and the constraints on their values. The last step is to define individuals or instances and fill in slot values. To represent the 'ontology' for machine processing and exchange of information among machines or agents, we are use the 'web ontology language' (OWL). OWL has capability for capturing the semantic richness of a defined 'ontology' [13]. We use UML to model the ontologies and the communication architecture. The semantic of 'ontology' modeling help generalize the common semantic thus hiding implementation details. The main aim of the use of UML notations is simply to create a graphical representation of 'ontologies' to make it easier for others to understand. Note that this use of UML notations to model the underlying 'ontology' should be distinguished from its use in software development to model the application domain.

The 'ontology' that is defined can cover the subject content of software engineering discipline which can be divided into five sub areas i.e. software process, requirements engineering, design, development, and verification and validation. In this paper we will not include the areas of project management and business domains which form our future work. For this example we developed an 'ontology-based' software engineering model which will promote a consistent understanding of software engineering across a team of multi-site software developers spread around the world and also serves as a common curriculum. As you may notice, software engineering knowledge already exits therefore we have simply re-structured it as an - 'ontology' – providing a common basis for communication questions problem solving solution development and exchange of information between the multi-site teams.

We not only define the terms but also the properties of these terms are. For example, the term of object class in UML can define attributes and operations of classes and relationship among classes. Therefore, properties of the term 'object class' can be class_name, class_attribute, class_operation, and class_relationship. Figure 1 illustrates 'object class' 'ontology' model, a meta-model of 'object

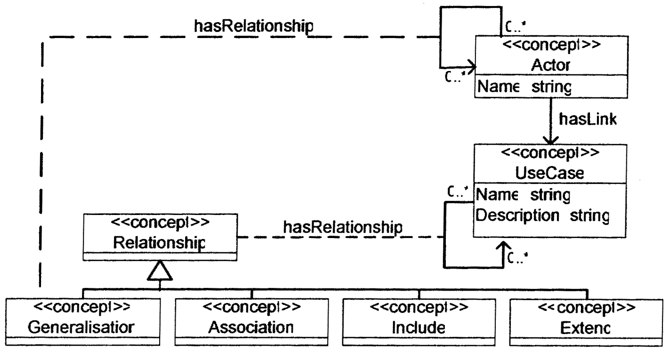

Figure 2: 'Use case ontology' model

class' model in UML. As you can see from Figure 1, the class_name property is Datatype property whose range can only be datatype value. Object property is distinguished from datatype property by whose range can be only instance. From the meta-model in Figure 1, object property is class_attribute, class_operation, and class_relationship.

A subclass-superclass hierarchy is A taxonomic relationship between a more general class and a more specific class. The more specific class is fully consistent with the more general element and contains additional information. For example, the term 'class relationship' in UML can be more specified by association, dependency, or generalization relationship. Association relationship can be better specified by aggregation or composition. Figure 1 shows the concept hierarchy of class relationship. Figure 2 depicts the 'use case ontology' model, a meta-model of the 'use case' model in UML. It shows relationships among actors and use cases within a system. 'Use case ontology' would be equally helpful as an alternative to the 'object class ontology'. Additionally there are five more ontologies that can be captured together with the object class diagram and 'use case' i.e. activity, state chart, package, sequence, and collaborative 'ontology'. These concepts are worth capturing in machine-interpretable manner.

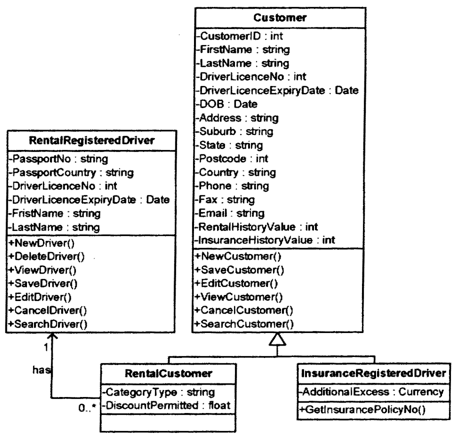

Figure 3: An example of class diagram put into the 'ontology' as instances

The last step of the development would be to add individuals as necessary. We have made an 'ontology' of software engineering and we now have a clear context to proceed. What we now need to do is to create data instances to be usable. Figure 3 shows an example of a class diagram for a particular project which we have used to map as instances of the 'object class ontology' model.

I am struggling to understand why we need it.

I think the system will be simpler for people to understand if we deleted the insurance registered driver.

My reasons for this are that the insurance registered driver is a sub type of the customer. This means that for every insurance registered driver object there must be a corresponding customer object. However, in the customer object we store values like customer type, insurance history value and rental history value. It does not make sense to have these values for the insurance registered driver. I also think people will be confused because we have the rental registered driver as an association with the rental customer (which is a sub type of the customer) but the insurance registered driver is a sub type of the customer.

Figure 4: An example of plain text communication.

V. 'ONTOLOGY-BASED' SOFTWARE ENGINEERING FOR MULTI-SITE SOFTWARE DEVELOPMENT

In this section we illustrate how 'ontology-based' software engineering facilitate multi-site distributed software development environment. Because there is no face to face communication, in multi-site distributed software development environments it is very difficult to develop a consistent understanding of the meaning of terminology or understanding of project agreement. Figure 4 shows a typical example of global communication which does not create consistent understanding of that particular project design. By using 'object class' 'ontology' of Figure 1 and instances of Figure 3 we are able to convert the plain text into a UML-like form that provides consistent understanding among multi-site developers. Software agents for example can be utilized to extract information from the 'ontology' described in OWL. To do so, software agent consults the 'object class ontology'. The 'ontology' shows how class is formed in the class diagram; each class contains a name, attributes, and operations; and relationships between the classes. Therefore, the software agent dynamically act to retrieve involved class name, involved class attributes, involved class operations, and involved relationships to draw a class diagram.

The 'ontology-based' software engineering is made explicit therefore becomes formalized in one way or another. Explanation of software engineering discipline is explicit (e.g. book form) but informal in knowledge. By using an 'ontology' we help to clarify the software engineering concepts thereby making them not only explicit but also formalizable and consistent for use by multi-site developers who would not have the opportunity to interact with each other to clarify meaning. The second advantage of the 'ontology' form is that it can be understood by machine-readable communication software.

VI. CONCLUSIONS AND FUTURE WORK

We have examined how 'ontology' can facilitate consistent understanding between multi-site developers. From the 'ontology' software repository, we can share components of software across multi-site distributed software development teams. Co-operation refers to the sharing of information and knowledge of the software production processes and how to support each other in the development process.

In future work we plan to try to expand our software development repository to help identify other components of collaboration such as communication, coordination, awareness, and interoperability by utilizing software agents.

VII. REFERENCES

[1] Capasso, R., Keeping Geographically Distributed Development Team in Sync. 2000, Rational Software.

[2] Audet, J. and G. Amboise, The Multi-Site Study: An Innovative Research Methodology. An online journal dedicated to qualitative research and critical inquiry, 2001. 6(2).

[3] Chang, Jayaratna, and Aisbett. Generational Change Model in the Software Life Cycle. in Proceeding of IASTED Conference on Software Engineering. 2003. Innsbruck, Austria.

[4] Bergland, G.D. A guided Tour of Program Design Methodologies. in IEEE Computer. 1981.

[5] Chang, Davis, and Chalup. A NEW LOOK AT THE ENTERPRISE INFORMATION SYSTEM LIFE CYCLE-Introducing the concept of generational change. in Proceeding of 5th International Conference on Enterprise Information Systems (ICEIS) 2003. 2003. Angers, France.

[6] Stuart, W. Evolutionary Project Management. in IEEE Computer. 1999.

[7] Davies, J., D. Fensel, and F.v. Harmelen, Towards the semantic web: 'ontology'-driven knowledge management. 2003: John Wiley & Sons.

[8] Sommerville, I., Software Engineering. 7th ed. 2004: Pearson Education Limited.

[9] Beck, K., Extreme Programming Explained. 2000: Addison Wesley.

[10] Wooldridge, M., Introduction to MultiAgent Systems. 1st ed. 2002: John Wiley & Sons.

[11] Chang, E. and T. Dillon. A Generation Model for Collaborated Distributed Software Development Paradigm. in To appear in Computer System Science and Engineering. 2003.

[12] Gruber, T.R. A translation approach to portable ontologies. in Knowledge Acquisition. 1993.

[13] McGuinness, D.L. and F.V. Harmelen, OWL Web 'ontology' Language Overview. 2004.