Meta-Model for Global Software Development to Support Portability and Interoperability in Global Software Development

Bugra Mehmet Yildiz, Bedir Tekinerdogan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
{bugra, bedir}@cs.bilkent.edu.tr

Abstract— **Global Software Development (GSD) considers the coordinated activity of software development that is not localized and central but geographically distributed. To support coordination among sites, usually it is aimed to adopt the same development and execution platform. Unfortunately, adopting a single platform might not be always possible due to technical or organizational constraints of the different sites in GSD projects. As such, very often GSD projects have to cope with portability and interoperability problems. To address these problems we propose to apply model-driven architecture design (MDA) approach. For this we present a common meta-model of GSD that we have derived from a systematic domain analysis process. The meta-model enhances the understanding of GSD, is used to define platform independent models of GSD architecture, and transform platform independent models to platform specific models.**

Keywords-Global Software Development, Architecture Modeling, Model-Driven Development

I. INTRODUCTION

Global Software Development (GSD) is a software development approach that can be considered as the coordinated activity of software development that is not localized and central but geographically distributed. In principle, GSD can be considered as the realization of outsourcing. The reason behind this globalization of software development stems from clear business goals such as reducing cost of development, solving local IT skills shortage, and supporting outsourcing and offshoring [1]. There is ample reason that these factors will be even stronger in the future, and as such, we will face a further globalization of software development [6].

One of the challenging issues in setting up global software development is the interoperability among the distributed sites [13][14]. Interoperability is defined as the ability of two or more systems or components to exchange information and to use the information that has been exchanged [7]. Although it is aimed to adopt the same platforms in global software development projects, this might not be always possible due to technical or organizational constraints. As such, different sites might run on different operating system platforms, use different component language platforms, or adopt a different middleware platform. Further, due to the continuous evolution of project requirements, the platforms on different sites might also need to evolve. Portability of the existing software to a new platform is not easy for even a single site development project; in the case of global software development projects this is even a much harder problem. Altogether, both the portability and interoperability problems will impede the adoption of a global software development approach.

Portability to different platforms and interoperability among different sites working on different platforms have been mainly addressed in the model-driven software development approaches. In this context, Model Driven Architecture (MDA) is a framework defined by the Object Management Group (OMG) that separates the platform specific concerns from platform independent concerns to improve the reusability, portability and interoperability of software systems [12]. To this end, MDA separates Platform Independent Models (PIMs) from Platform Specific Models (PSMs). The PIM is a model that abstracts from any implementation technology or platform. The PIM is transformed into one or more PSMs, which include the platform specific details. Finally the PSM is transformed to code providing the implementation details. Obviously by separating the platform specific concerns and providing mechanisms to compose these concerns afterwards in the code MDA provides a clean separation of concerns and as such the systems are better reusable easier to port to different platforms and have increased interoperability.

We present the model-transformation pattern for transforming the global platform independent model to the local platform specific models. An important part of the model transformation is the common GSD meta-model. We describe both the abstract syntax and the concrete syntax of the meta-model. The abstract syntax is defined using the UML notation; the concrete syntax is specific for the parts of the meta-model. The meta-model enhances the understanding of GSD, and supports the model transformation for solving portability and interoperability problems.

The remainder of the paper is structured as follows. Section II provides some background on GSD. Section III describes the meta-model for GSD and Section IV describes the related work. Finally, Section VI concludes the paper.

II. GLOBAL SOFTWARE DEVELOPMENT

A GSD architecture usually consists of several nodes, or sites, on which different teams are working to develop a part of the system. The teams could include development teams, testing team, management team, etc. Usually, each site will also be responsible for following a particular process. In addition, each site might have its own local data storage. Overall we can identify four important key concerns in designing GSD:

Development - the software development activities typically using a software development process. This includes activities such as requirements analysis, design, implementation and testing. Each PDS will address typically a subset of these activities.

Communication – communication mechanisms within and across sites. Typically the different sites need to adopt a common communication protocol.

Coordination – coordination of the activities within and across sites to develop the software according to the requirements. Coordination will be necessary to align the workflows and schedules of the different sites. An important goal could be to optimize the development using appropriate coordination mechanisms.

Control – systematic control mechanisms for analyzing, monitoring and guiding the development activities. This does not only include controlling whether the functional requirements are performed but also which and to what extent quality requirements are addressed.

In fact each of these concerns requires further in-depth investigation and has also been broadly discussed in the GSD community.

To realize multi-site development is not a trivial task. In particular if the different sites are working on different platforms the interoperability problems must be resolved.

Figure 1 shows the transformation pattern for mapping a global platform independent model to local platform specific models. The platform independent model can be considered the same across multiple development sites. If needed the local sites can keep working on different platforms. In that case the alignment and the interoperability can be achieved by defining transformation patterns, which map the local platform models to the global platform independent models, and vice versa. To support the model transformation a proper definition of the GSD meta-model is necessary. We discuss this in the next section.

.

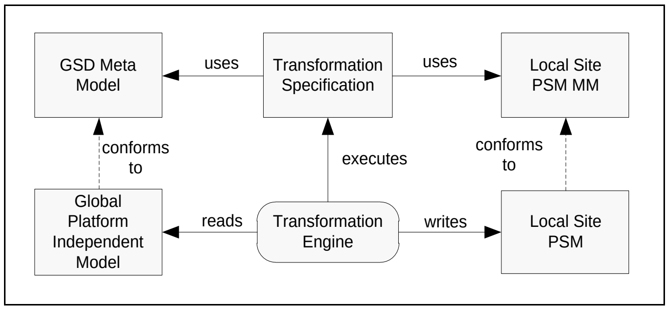

Figure 1. Model-Transformation pattern for mapping GSD PIM to local PSM

III. META-MODEL FOR GLOBAL SOFTWARE DEVELOPMENT

Meta-models define the language for the models. In both software language engineering [9] and model-driven development domains [2], a meta-model should have the following two key elements:

Abstract Syntax: Captures the concepts provided by the language and relationships between these concepts.

Concrete Syntax: Defines the notation that facilitates the presentation and construction of models in that language.

Based on the literature of GSD, we have defined a meta-model for GSD that defines the concepts and their relations to enhance the understanding of GSD and support the model transformation. Since the meta-model is quite large and we aim the modeling of different concerns of GSD, we have decomposed meta-model into six meta-model units. Each of these meta-model units includes semantically close entities and address different concerns. These units are *Deployment*, *Process*, *Data*, *Communication*, *Tool* and *Migration.* Each unit includes abstract syntax representing GSD elements and their relations and visual concrete syntax for visualization of these elements.

A. Deployment Unit

Deployment Unit concerns the deployment of the teams to different sites. The abstract and concrete syntax of this unit are shown in Figure 2.

Team is the primary essential entity in Deployment and also in the whole meta-model and is defined as a group of persons that work together to achieve a particular goal. A *Team* may be organized in a temporary way that it will be dismissed after its function is complete. *Team* is allocated at a particular *Site*. *Site* may to a country, city or a building where a Team works at. Location attribute determines where *Site* is placed in the world. Time zone shows the local time of *Site*. *Teams* may belong to different types of *Organizations*, such as commercial organizations, subcontractors or non-profitable organizations such as open source communities. *Teams* can be from different countries and depending on the society they are in, they may have different *Social Cultures*. Like *Social Culture*, *Team*'s background including work experience, the time that members work together, their habits are captured by *Work Culture* entity. *Expertise Area*, *Team* and *Site* can be further decomposed into sub-parts. For example, a Software Team may consist of sub-Teams each responsible for Design, Implementation, Testing and Integration.

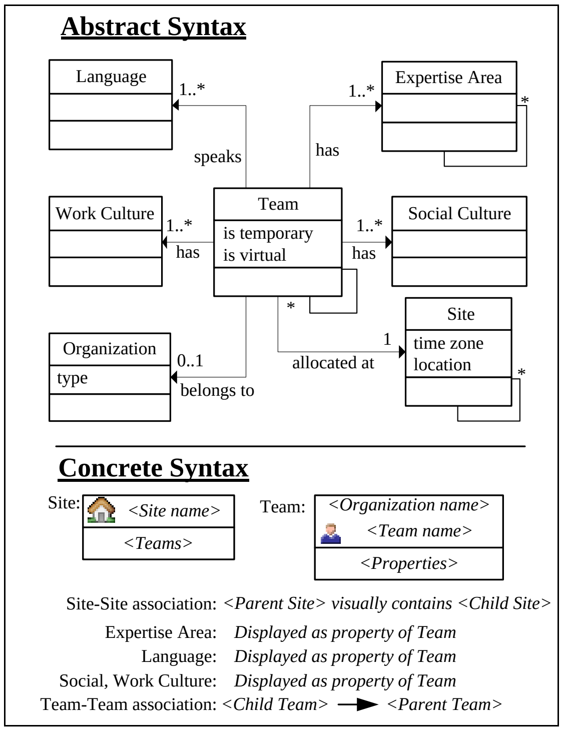
Abstract Syntax
Language
Expertise Area
Work Culture
Team
is temporary
is virtual
Social Culture
Organization
type
Site
time zone
location
speaks
has
belongs to
allocated at
1..*
0..1
1
*
Concrete Syntax
Site: <Site name>
<Teams>
Team: <Organization name>
<Team name>
<Properties>
Site-Site association: <Parent Site> visually contains <Child Site>
Expertise Area: Displayed as property of Team
Language: Displayed as property of Team
Social, Work Culture: Displayed as property of Team
Team-Team association: <Child Team> <Parent Team>

Figure 2. Deployment Unit: Abstract and Concrete Syntax

B. *Process Unit*

Process Unit concerns the different kind of processes in GSD. The abstract and concrete syntax of this unit are shown in Figure 4.

Process is defined as a planned set of activities that aims to provide some service. *Teams* participate in *Process* in order to provide some service. Service is defined with *Function*. A *Function* can be any service during software development process that requires some *Expertise Areas* such as software development, architecture design, business management, requirements elicitation and so on. *Coordination* is also a *Function* that should be provided for coordinating several *Teams*' activities. A *Process* consumes or uses several different *Data Entities* and also creates other *Data Entities* for providing targeted *Function*s. For supporting activities defined in *Process*, *Process* concept is further specialized into Workflow, Business Process and Development Process (not shown in figure).

C. *Data Unit*

Data Unit is for representing ownership and physical deployment of software development data. The abstract and concrete syntaxes are shown in Figure 4.

Data Entity is the fundamental entity of this unit. It represents any piece of data: digital, textual or informal piece of information such as notes taken by developers, telephone calls that are usually not recorded. *Data Entity* has size whose unit is defined by size type; for example, a 120-page report, 6 minutes of voice record, 2 gigabyte of digital data. *Creation date* and *last update date* show the history of *Data Entity*. *Data Entity* has *Actual Type* where *Actual Format* can be one of predefined formats (video, sound, text, picture and complex-*Data Entity*) or some designer defined format. If *Data Entity* is digital, then in addition to *Actual Format*, it has a *Digital Format*. *Data Entity* may be implemented in one or more *Languages.*

Data Entity is stored in *Data Storage*. *Data Storage* corresponds to any object in real world that can store information. For example, some textual document is stored in paper form, or it is stored in a voice record, or it is stored digitally in the format of some text editor. *Data Storage* has ability to store some *Actual Types* and if it can store digital data, then it can support some *Digital Types* also. A *Data Storage* instance is owned by one or more *Teams* and it can be located in one *Site* or may be distributed over several *Sites* like distributed databases.

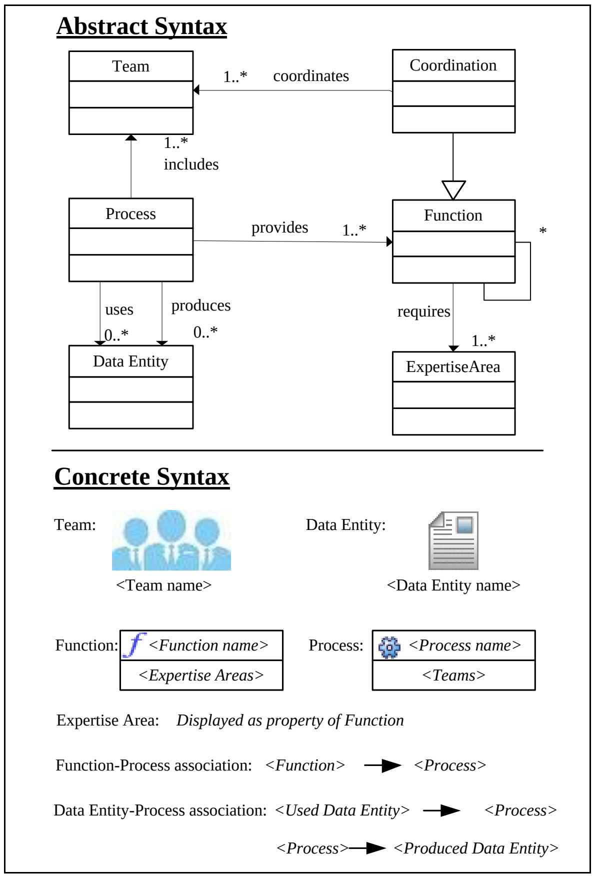

Figure 3. Process Unit: Abstract and Concrete Syntax

D. *Communication Unit*

Communication Unit focuses on the representation of both formal and informal communication activities between *Teams*. The abstract and concrete syntaxes are shown in Figure 5.

Communication is done over *Communication Platform* in the context of *Process* and it can be an instance of sudden/event based communication activity like a telephone call or a continuous communication channel such as a discussion forum. *Type* attribute is for representing in which way *Communication* takes place such as email, phone call, face-to-face chat and so on. *Suggested time period* is an important attribute for GSD since Teams work in different time zones, some *Communication* channels can be used effectively in a defined time period. For example, phone calls should be done during the hours when both sides are in or around their work hours.

Communication has two sides, which are caller and receiver. Generally speaking, caller starts communication and receiver is the one who is called by caller. For example, an email sender is classified as caller and receiver is the one who receives email. Sometimes, there can be multiple callers such as video conferences or there can be multiple receivers such as discussion forums. It is also possible that caller and receiver are the same such as a planned meeting. For all cases, caller and receivers are considered as *Teams* in this unit. While *Teams* communicate, one or more *Data Entities* are carried in the context of *Communication*.

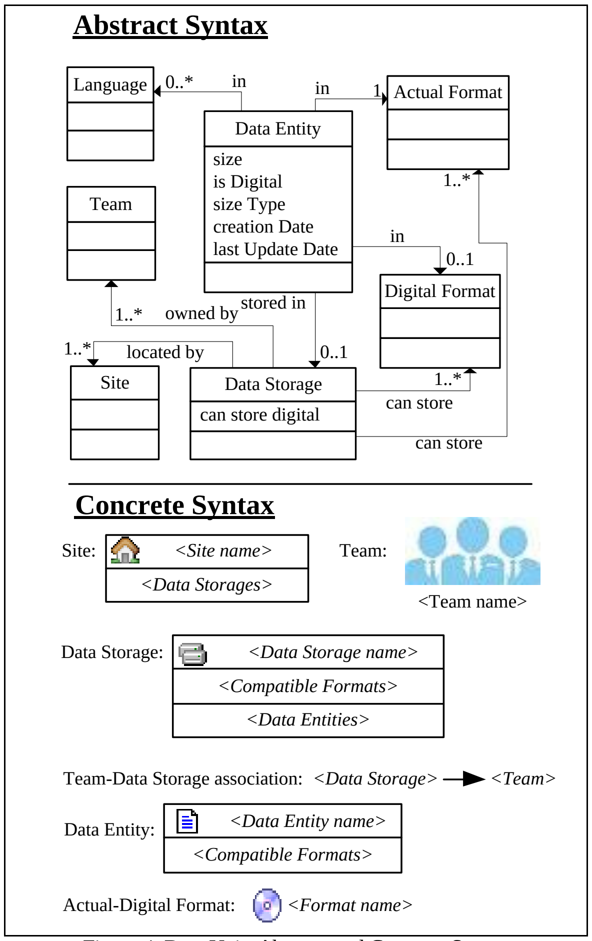

Figure 4. Data Unit: Abstract and Concrete Syntax

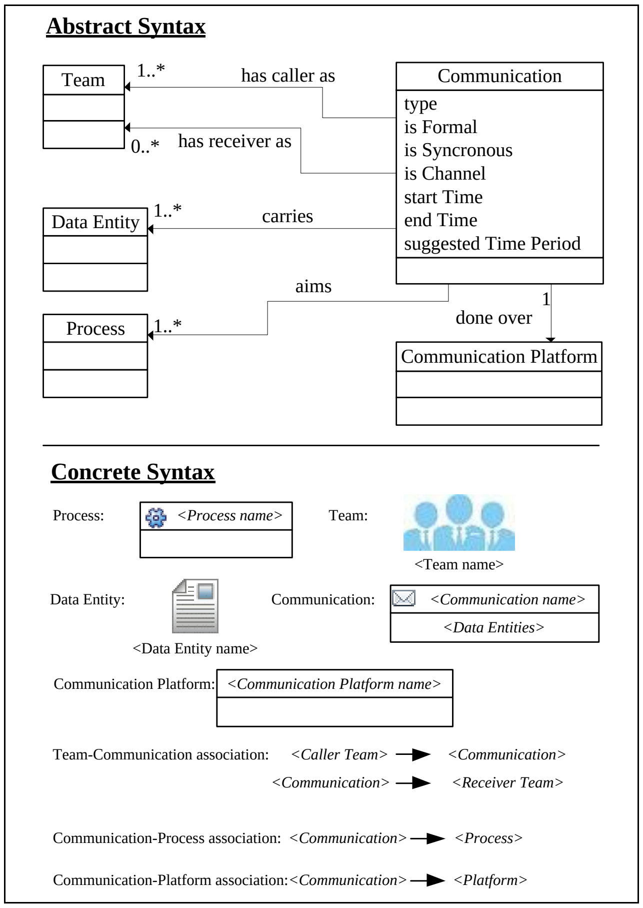

Figure 5. Communication Unit: Abstract and Concrete Syntax

E. *Tool Unit*

Tool Unit captures details of tools used by *Teams* for communication and providing *Functions*. The abstract and concrete syntax are shown in Figure 6.

Tool is compatible with one or more Actual Format and Digital Format. Platform is the set of *Tools* used by *Teams* for communication or providing some functions. Depending on the purpose, the platform is defined as *Function Platform* or *Communication Platform*.

F. *Migration Unit*

Migration Unit concerns the migration and traveling of *Teams* during GSD activities. These travels are especially needed in the first and final phases of the projects to ease and support coordination and integration. The abstract and concrete syntax are shown in Figure 6.

Migration is executed by one or more *Teams* from *Site* to *Site* at a particular date. In a *Migration*, *Teams* may carry *Data Storage* such as documents, digital data containers and so on. *Migration* is executed in the context of *Process*.

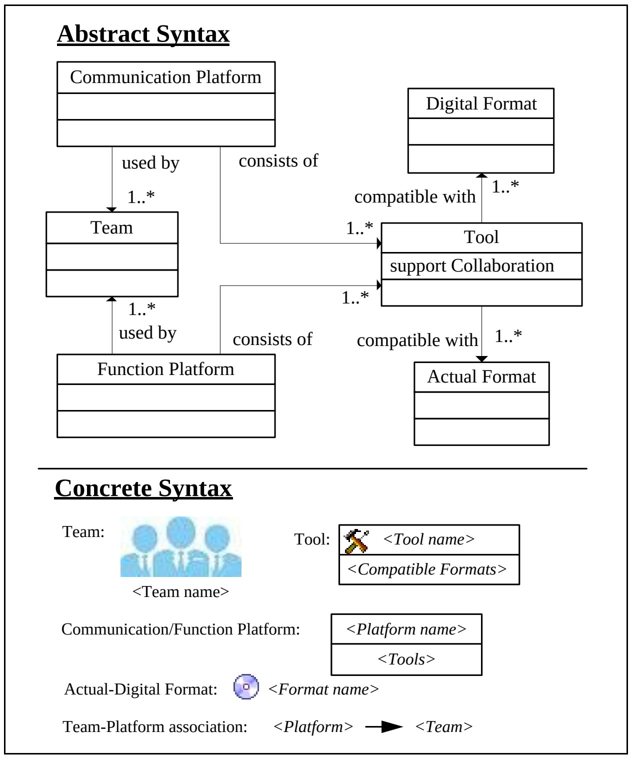

Figure 6. Tool Unit: Abstract and Concrete Syntax

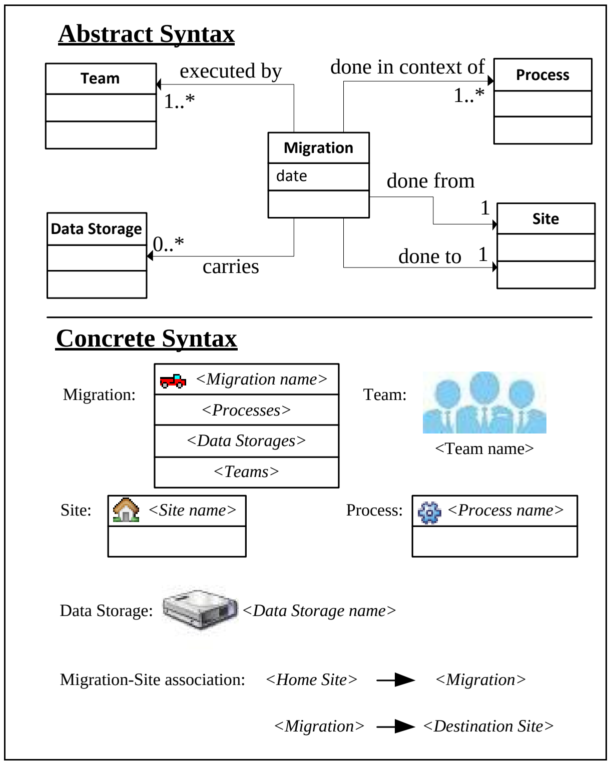

Figure 7. Migration Unit: Abstract and Concrete Syntax

G. *Example Case*

As an example case, consider a GSD environment with 5 Sites. Company A operates in United States. Customer relations and requirements management jobs are done in New York while software architecture is designed in Los Angeles. Company B is hired as subcontractor for developing software and testing, which is located in Pekin, China. Moving from this case definition and Deployment meta-model unit, the model in Figure 9 can be drawn.

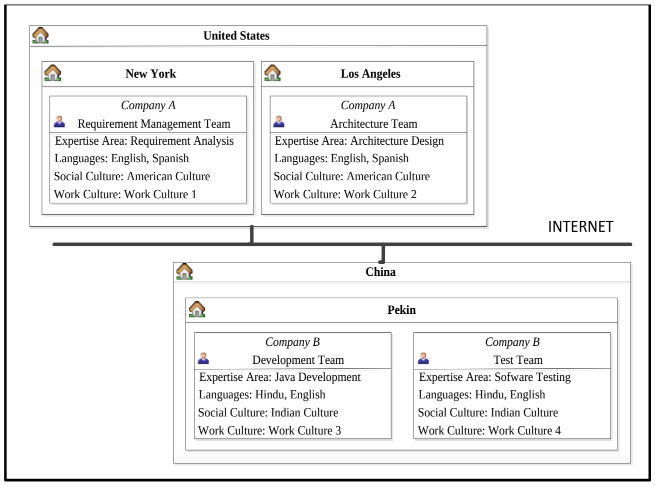
United States
New York
Company A
Requirement Management Team
Expertise Area: Requirement Analysis
Languages: English, Spanish
Social Culture: American Culture
Work Culture: Work Culture 1
Los Angeles
Company A
Architecture Team
Expertise Area: Architecture Design
Languages: English, Spanish
Social Culture: American Culture
Work Culture: Work Culture 2
INTERNET
China
Pekin
Company B
Development Team
Expertise Area: Java Development
Languages: Hindu, English
Social Culture: Indian Culture
Work Culture: Work Culture 3
Company B
Test Team
Expertise Area: Sofware Testing
Languages: Hindu, English
Social Culture: Indian Culture
Work Culture: Work Culture 4

Figure 9. Example Case Model

IV. Related Work

Notably, architecting in GSD has not been widely addressed. The key research focus in the GSD community seems to have been in particular related to tackling the problems related to communication, coordination and control concerns. Clerk et al. [4] report on the use of so-called architectural rules to tackle the GSD concerns. Architectural rules are defined as “principles and statements about the software architecture that must be complied with throughout the organization”. They have defined four challenges in GSD: time difference and geographical distance, culture, team communication and collaboration, and work distribution. For each of these challenges they list possible solutions and describe to what extent these solutions can be expressed as architectural rules. The work of Clerk et al. aims to shed light on what kind of architectural rules are necessary to guide the GSD. We consider our work complementary to this work. In our work the design actions that relate to the expected answers of questions are defined as design actions.

Tool support has been named as one of the important challenges for GSD since it requires making software development tools and environments more collaborative [13]. Booch and Brown [3] have introduced the vision for Collaborative Development Environment (CDE), which is defined as “a virtual space wherein all the stakeholders of the project – even if distributed by time or distance – may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to create an executable deliverable and its supporting artifacts". A number of efforts have been carried out to support the idea of CDEs. Whitehead [13] has presented a survey on existing collaboration support tools in software engineering. Whitehead distinguishes among four broad categories of tool support to support collaboration in software engineering: Model-based collaboration tools for representing the adopted models; Process support tools for representing software development process; Awareness tools for informing developers about the ongoing work of others and to avoid conflicts; Collaboration infrastructure to support data and control integration and likewise support interoperability. Despite the clear need and benefits of the existing CDE tools, it appears that most of the work on CDE has focused on the (social) collaboration concern and less on the (technical) development part. Further the tools that address development primarily focus on collaborative coding and relatively little attention has been paid to architecture design. There seems to be a general agreement that more research is needed in this domain. Our approach and the meta-model definition can be considered as part of the efforts for enhancing CDE for design of GSDs.

Maciel et al. [10] present a domain-specific architecture (DSA) defining middleware services to provide interoperability in collaborative environments. Similar to our approach they define a platform independent model that is independent of platform specific models. In their approach the reference architecture (PIM) is based on MDA's UML Profile for Enterprise Distributed Object Computing (EDOC) [11] and the viewpoints defined in RM-ODP (Open Distributed Processing-Reference Model) are adopted [8]. In our approach we do not use a general purpose architecture framework such as RM-ODP but adopt a meta-model based on a domain analysis of the GSD literature.

V. CONCLUSION AND FUTURE WORK

Different challenges have been identified to set up a Global Software Development environment. Our literature study on GSD showed that in particular the challenges of communication, coordination, and control of GSD is addressed in the GSD community but less focus has been provided on the modeling, documentation and analysis of architecture for GSD. One of the key technical problems in GSD projects is the evolution of platforms on different sites and the need for interoperability among different sites. A close analysis of the literature shows that the application of MDSD has not been explicitly addressed, neither in the GSD community nor in the MDSD community. In this paper we have provided a general transformation pattern for mapping a global platform independent model to the platform specific models at local sites. Portability can be supported by defining transformation definition that map the new platform models to the global platform independent models and vice versa. Interoperability is supported due to the common model, global platform independent model that conforms to the meta-model that we have defined in the paper. The meta-model aimed to support the portability and interoperability in GSD but also enhances the understandability and communication about GSD. In our future work we plan to define domain specific languages for the six units of the GSD meta-model. For this we will use the Eclipse Modeling Framework [5] and develop the corresponding tool support for realizing the automatic or semi-automatic model transformations in GSD projects.

REFERENCES

[1] P. J. Agerfalk, B. Fitzgerald, H. H. Olsson, and E. O´ Conchu´ir, "Benefits of Global Software Development: The Known and Unknown," in International Conference on Software Process, ICSP 2008. 2008. Leipzig, Germany,: Springer Berlin / Heidelberg

[2] J. Bézivin. On the Unification Power of Models. Software and System Modeling (SoSym) 4(2):171-188, 2005.

[3] G. Booch and A. Brown. Collaborative Development Environments. Advances in Computers Vol. 59, Academic Press, August, 2003.

[4] V. Clerc, P. Lago and H. van Vliet, "Global Software Development: Are Architectural Rules the Answer?" Proc. of the 2nd International Conference on Global Software Engineering, pp. 225–234. IEEE Computer Society Press, Los Alamitos, 2007.

[5] Eclipse Modeling Framework, http://www.eclipse.org/ modeling/emf/, accessed: July 2011.

[6] J. D. Herbsleb, Global Software Engineering: The Future of Socio-technical Coordination, 2007 Future of Software Engineering, p.188-198, May 23-25, 2007

[7] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[8] ISO-2004. Use of UML for ODP system specification. Working Draft. ISSO/IEC JTC1/SC7.

[9] A. Kleppe. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Longman Publishing Co., Inc., Boston, 2009.

[10] R. S. P. Maciel, C. G. Ferraz, and N. S. Rosa, "An MDA domain specific architecture to provide interoperability among collaborative environments," in Proceedings of the 19th Brazilian Symposium on Software Engineering (SBES '05), pp. 1–16, Uberlandia, Brazil, October 2005.

[11] OMG EDOC. UML Profile for Enterprise Distributed Object Computing Specification. OMG Adopted Specification (ptc/02-02-05), 2002.

[12] OMG Model Driven Architecture, OMG Model Driven Architecture. http://www.omg.org/mda/. Accessed in June 1,2011.

[13] J. Whitehead, Collaboration in Software Engineering: A Roadmap, In FOSE '07: 2007 Future of Software Engineering, pp. 214-225, 2007.

[14] I. S. Wiese and E. H. M. Huzita, "IMART: An Interoperability Model for Artifacts of Distributed Software Development Environments," Global Software Engineering, 2006. ICGSE '06. International Conference on , vol., no., pp. 255-256, Oct. 2006.