2011 Sixth IEEE International Conference on Global Software Engineering Workshops

Architectural Viewpoints for Global Software Development

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

Abstract— **Global Software Development (GSD) can be considered as the coordinated activity of software development that is not localized and central but geographically distributed. Designing an appropriate software architecture of a GSD system is important to meet the requirements for the communication, coordination and control of the distributed GSD teams. A common practice in software architecture design is to apply architectural views to model the design decisions for the various stakeholder concerns. Unfortunately, existing architectural viewpoint approaches are general-purpose and not directly dedicated towards GSD projects. In this paper we propose six architectural viewpoints that have been specifically defined to model GSD systems. The architectural viewpoints are based on a meta-model that has been derived after a thorough domain analysis of the GSD literature.**

Keywords-Global Software Development, Architecture Modeling, Architectural Viewpoint

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 [8]. 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 [8].

Designing an appropriate software architecture of a GSD system is important to meet the requirements for the communication, coordination and control of the distributed GSD teams. Software architecture forms one of the key artifacts in the entire software development life cycle since it embodies the earliest design decisions and includes the gross-level components that directly impact the subsequent analysis, design and implementation [5]. A common practice in the software architecture design community is to model and document different architectural views for describing the architecture according to the stakeholders' concerns [5][9][12][13]. An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern. Having multiple views helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Architectural views conform to viewpoints that represent the conventions for constructing and using a view. An architectural framework organizes and structures the proposed architectural viewpoints. Different architectural frameworks have been proposed in the literature. Examples of architectural frameworks include the Kruchten's 4+1 view model [11], the Siemens Four View Model [9] , and the Views and Beyond approach (V&B) [5]. In general the existing architectural frameworks tend to be general purpose and not directly focused to a particular domain. The advantage of this is that it can be applied in a broad set of domains, but on the other hand the general-purpose architectural frameworks can fall short for modeling the particular concerns of specific domains.

In this paper we propose an architectural framework including six architectural viewpoints which have been specifically defined for modeling GSD architecture. The architectural framework is based on a meta-model of GSD that we have defined after a thorough domain analysis of the related GSD literature. The meta-model consists of six different parts which each represent the key concepts, i.e. abstract syntax, of the corresponding architectural viewpoint. In addition we have defined the required notation, concrete syntax, that can be used in each architectural viewpoint.

The remainder of the paper is structured as follows. Section II describes the key concerns for architecting GSD and domain specific languages. Section III describes the viewpoints for GSD. Section IV provides a short example illustrating the usage of some GSD viewpoints. Section V describes the related work and finally section VI concludes the paper.

II. GLOBAL SOFTWARE DEVELOPMENT

Figure 1 shows the conceptual architecture for Global Software Development systems. 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.

978-0-7695-4558-5/11 $26.00 © 2011 IEEE
DOI 10.1109/ICGSE-W.2011.11

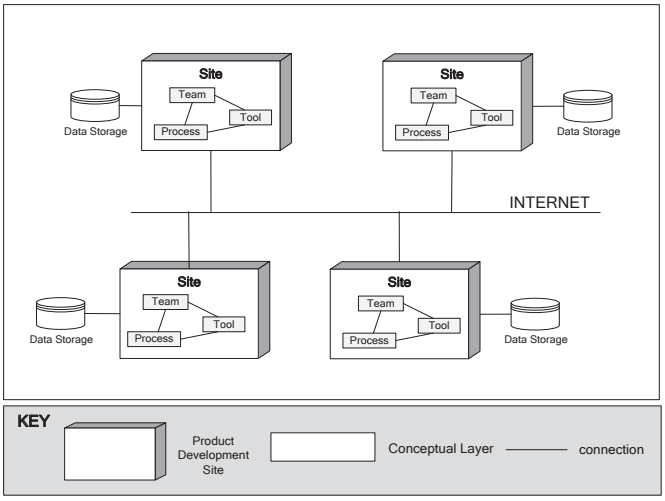

Figure 1. Conceptual Architecture for GSD

Overall the following four important key concerns are identified for 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 site will address typically a subset of these activities.

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

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. The important issue that we would like to address is that each of these concerns can be realized in different ways and likewise will shape the GSD architecture. The GSD software architect needs to identify and define these concerns to design and document the GSD architecture accordingly. To communicate the design decisions, guide the development process and analyze the GSD architecture it is important to model the GSD architecture properly. Unfortunately, each of these concerns are specific to GSD and are not explicitly addressed by existing architectural viewpoint approaches. In the following section we propose the viewpoints for GSD that aim to address the above and other GSD concerns.

III. GLOBAL SOFTWARE DEVELOPMENT VIEWPOINTS

Defining a new architectural viewpoint implies writing a *viewpoint guide*. This is similar to the notion of *style guide* as defined in [5]. The *viewpoint guide* defines the vocabulary of the architectural element and relation types, and defines the rules for how that vocabulary can be used. For defining a viewpoint guide for a particular quality concern we apply the template as defined in Table 1.

TABLE 1. VIEWPOINT GUIDE TEMPLATE FOR GSD

Viewpoint Element	Description
Name	Unique name for the viewpoint
Element Types	The architectural element types native to the viewpoint
Relation Types	The relation types among architectural elements
Properties of Elements	Additional information on the element types
Topology Constraints	The rules of composition of the elements and relations.
Notation	The adopted notation for the element types and relation types. The notation can be textual or visual.
Relation to other views/viewpoints	The relation to other viewpoints other than the base viewpoint

A viewpoint defines the template for the views that can be instantiated from it. We consider viewpoints as meta-models representing the basic concepts of the architecture from a particular perspective. To define viewpoints and likewise the viewpoint template, we believe it is necessary to define the corresponding meta-model. Meta-models define the language for the models. In both software language engineering [10] 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.

Given these key elements of a language, we can also evaluate viewpoints of Global Software Development as the languages for defining GSD views.

Likewise to define the viewpoints for GSD we have to define the corresponding meta-models including both the abstract syntax and the concrete syntax. Based on the literature of GSD, we have defined a meta-model for GSD that defines the concepts and their relations to derive application architecture from different viewpoints. The meta-model consists of six different parts each representing an architectural viewpoint for GSD systems. The viewpoints that we have defined based on the meta-model are *Deployment*, *Process*, *Data*, *Communication*, *Tool* and *Migration Viewpoints.* These viewpoints are described in more detail in the following subsections. For each viewpoint we first provide the abstract syntax, and then the corresponding notation, concrete syntax.

A. Deployment Viewpoint

Deployment Viewpoint concerns the deployment of the teams to different sites. The abstract syntax of this viewpoint is shown in Figure 2. Viewpoint guide for Deployment Viewpoint is shown in Table 2.

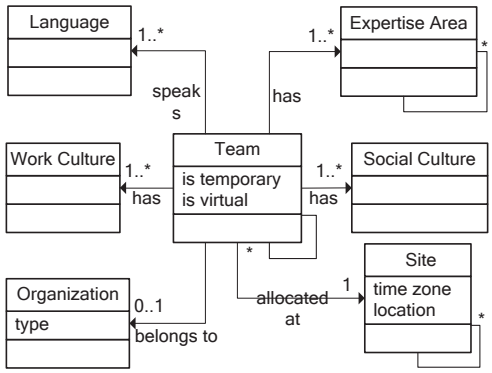

Figure 2. Deployment Viewpoint: Abstract Concrete Syntax

TABLE 2. DEPLOYMENT VIEWPOINT FOR GSD

Viewpoint Element	Description
Name	Deployment Viewpoint
Element Types	Team, Site, Organization, Language, Expertise Area, Social Culture, Work Culture
Relation Types	allocated-at, belongs-to, has, speaks, parent-child
Properties of Elements	All elements: name, description, id Team: isTemporary, isVirtual Site: timezone, location Organization: type
Topology Constraints	- Team hierarchy must be in the structure of tree. - Expertise Area hierarchy must be in the structure of tree. - Site hierarchy must be in the structure of tree.
Notation	Site: <Site name> <Teams> Team: <Organization name> <Team name> <Properties> Language: as property of Team Expertise Area: as property of Team Social Culture: as property of Team Work Culture: as property of Team
Relation to other views/viewpoints	- Team entity is common for all viewpoints. - Expertise Area entity is used in Process viewpoint.

Team is the primary essential entity in the viewpoint and is defined as a group of persons that work together to achieve a particular goal. *Team* may be organized in a temporary way and be dismissed after its function is complete. *Team* is allocated at a particular *Site*. *Site* may be located in a country, city or a building where a Team works at. The attribute *location* 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*. *Team* may further include *Work Culture* including work experience, the time that members work together, their habits etc. *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.

B. Process Viewpoint

Process Viewpoint concerns the different kind of processes in GSD. The abstract syntax of Process unit is shown in Fig 3. Viewpoint guide for this viewpoint is shown in Table 3.

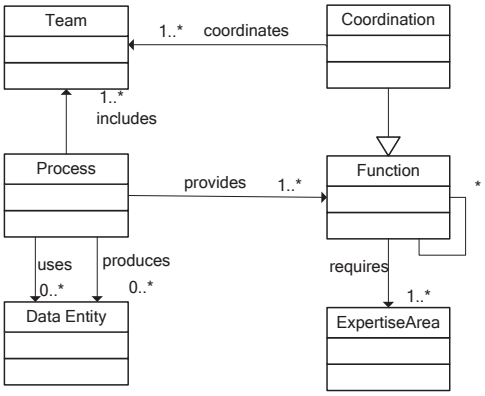

Figure 3. Process Viewpoint: Abstract Syntax

TABLE 3. PROCESS VIEWPOINT FOR GSD

Viewpoint Element	Description
Name	Process Viewpoint
Element Types	Team, Coordination, Process, Function, Data Entity, Expertise Area
Relation Types	coordinates, includes, provides, uses, produces, requires, parent-child, extends
Properties of Elements	All elements: name, description, id
Topology Constraints	- Function hierarchy must be in the structure of tree.
Notation	Data Entity: <Data Entity name> Team: <Team name> Function: <Function name> <Expertise Areas> Process: <Process name> <Teams> Function-Process association: <Function> → <Process> Data Entity-Process association: <Used Data Entity> → <Process> <Process> → <Produced Data Entity> Expertise Area: as property of Function
Relation to other views/viewpoints	- Team entity is common for all viewpoints. - Expertise Area entity is used in Deployment viewpoint. - Data Entity entity is used in Data viewpoint.

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 *Functions*. For supporting activities defined in *Process*, *Process* concept is further specialized into Workflow, Business Process and Development Process (not shown in figure).

C. Data Viewpoint

Data Viewpoint is for representing ownership and physical deployment of software development data. The abstract syntax is shown in Figure 4. Table 3 shows the viewpoint guide for Data Viewpoint.

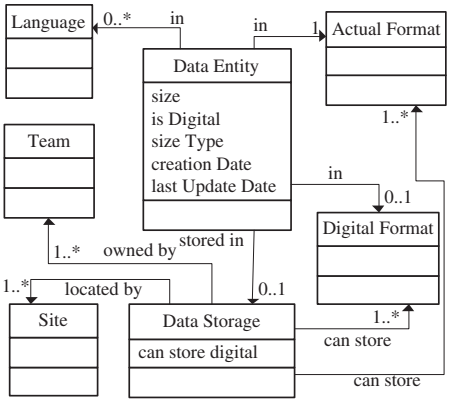

Figure 4. Data Viewpoint: Abstract Syntax

Data Entity is the fundamental entity of this viewpoint. 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.

TABLE 4. DATA VIEWPOINT FOR GSD

Viewpoint Element	Description
Name	Data Viewpoint
Element Types	Language, Team, Site, Data Storage, Digital Format, Actual Format, Data Entity
Relation Types	in, owned by, stored in, located by, can store
Properties of Elements	All elements: name, description, id Data Storage: canStoreDigital Data Entity: size, isDigital, sizeType, creationDate, lastUpdateDate
Topology Constraints	- Data Storage can store Digital Format only if Data Storage can store digital. - Data Entity is in some Digital Format only if Data Entity is digital.
Notation	Site: <Site name> <Data Storages> Team: <Team name> Data Storage: <Data Storage name> <Compatible Formats> <Data Entities> Data Entity: <Data Entity name> <Compatible Formats> Actual-Digital Format: <Format name> Team-Data Storage association: <Data Storage> → <Team>
Relation to other views/viewpoints	- Team entity is common for all viewpoints. - Data Entity entity is used Process and Communication viewpoints. - Data Storage is used in Migration viewpoint. - Actual and Digital Formats are used in Tool Viewpoint.

D. Communication Viewpoint

Communication Viewpoint focuses on the representation of both formal and informal communication activities between *Teams*. The abstract and concrete syntax are shown in Figure 5. Table 5 displays the viewpoint guide.

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.

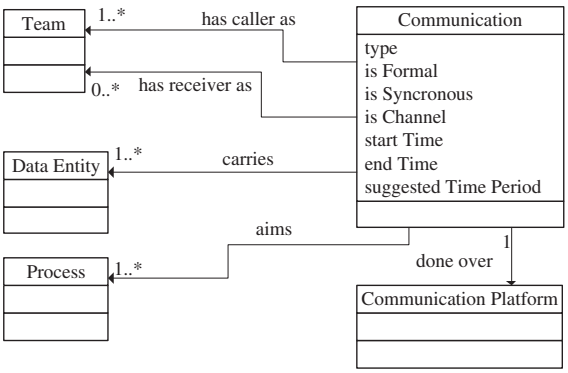

Figure 5. Communication Viewpoint: Abstract Syntax

TABLE 5. COMMUNICATION VIEWPOINT FOR GSD

Viewpoint Element	Description
Name	Communication Viewpoint
Element Types	Team, Communication, Process, Data Entity, Communication Platform
Relation Types	has caller as, has receiver as, carries, aims, done over
Properties of Elements	All elements: name, description, id
Topology Constraints	- None.
Notation	Data Entity: <Data Entity name> Team: <Team name> Process: <Process name> / <Teams> Communication: <Communication name> / <Data Entities> Communication Platform: <Communication Platform name> Team-Communication association: <Caller Team> → <Communication>, <Communication> → <Receiver Team> Communication-Process association: <Communication> → <Process> Communication-Platform association: <Communication> → <Platform>
Relation to other views/viewpoints	- Team entity is common for all viewpoints. - Process entity is used in Process viewpoint. - Data Entity entity is used in Data viewpoint.

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 viewpoint. While *Teams* communicate, one or more *Data Entities* are carried in the context of *Communication*.

E. Tool Viewpoint

Tool Viewpoint captures details of tools used by *Teams* for communication and providing *Functions*. The abstract syntax is shown in Figure 6 and viewpoint guide is shown .n Table 5.

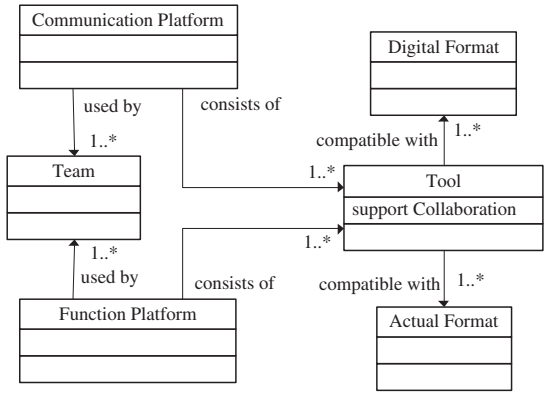

Figure 6. Tool Viewpoint: Abstract Syntax

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*.

TABLE 6. TOOL VIEWPOINT FOR GSD

Viewpoint Element	Description
Name	Tool Viewpoint
Element Types	Communication Platform, Function Platform, Team, Digital Format, Actual Format, Tool
Relation Types	used by, consists of, compatible with
Properties of Elements	All elements: name, description, id Tool: supportCollaboration
Topology Constraints	- None.
Notation	Tool: <Tool name> / <Compatible Formats> Team: <Team name> Communication/Function Platform: <Platform name> / <Tools> Actual-Digital Format: <Format name> Team-Platform association: <Platform> → <Team>
Relation to other views/viewpoints	- Team entity is common for all viewpoints. - Actual and Digital Formats are used in Data Viewpoint. - Communication Platform is used in Communication Viewpoint.

F. Migration Viewpoint

Migration Viewpoint 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. Figure 7 shows the abstract syntax of Migration Viewpoint. Viewpoint guide for Migration Viewpoint is shown in Table 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*.

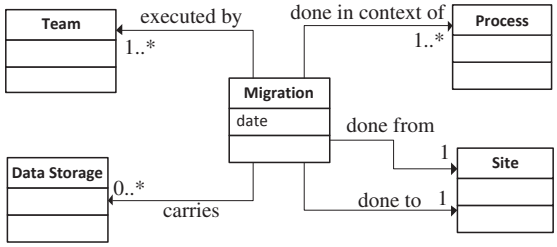

Figure 7. Migration Viewpoint: Abstract Syntax

TABLE 7. MIGRATION VIEWPOINT FOR GSD

Viewpoint Element	Description
Name	Migration Viewpoint
Element Types	Team, Process, Data Storage, Site, Migration
Relation Types	executed by, done in context of, done from/to, carries
Properties of Elements	All elements: name, description, id Migration: date
Topology Constraints	- Source and destination Sites for a Migration can't be the same.
Notation	Site: <Site name> Team: <Team name> Process: <Process name> Migration: <Migration name>, <Processes>, <Data Storages>, <Teams> Data Storage: <Data Storage name> Migration-Site association: <Home Site> → <Migration> <Migration> → <Destination Site>
Relation to other views/viewpoints	- Team entity is common for all viewpoints. - Process is used in Process viewpoint. - Data Storage is used in Data Storage viewpoint.

IV. EXAMPLE CASE

As an example case, consider a GSD project with 7 Sites located at different places in the world. Company A, the owner of the project, has its center operating in United States. Requirement Analysis is done in New York and Architecture Team works in Silicon Valley, California. Company A has development center in New Delhi, India where software development and test Teams work. Also, Company A works with a subcontractor company, Company B for some particular components. Company B's center is located in Sao Paulo, Brazil. Company B also has development and test Teams. Figure 8 shows the Deployment View of this case description. The Deployment View is instantiated from the Deployment Viewpoint in Table 2.

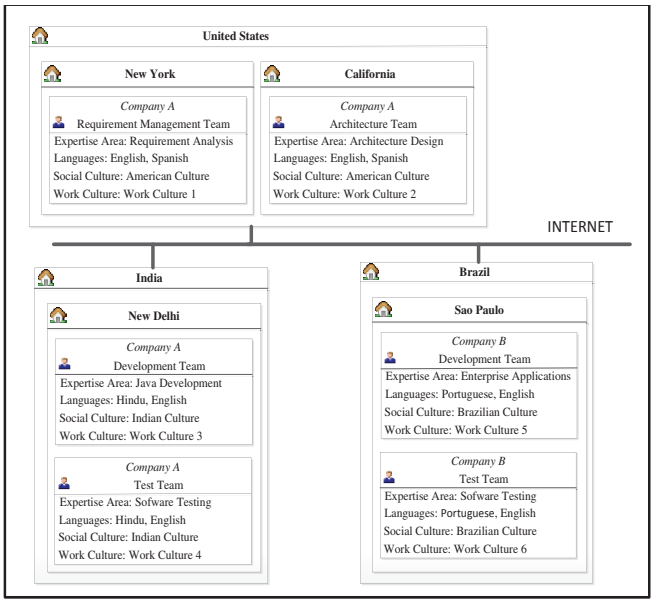

Figure 8. Example Case: Deployment View

For the example case described in the previous paragraph, we also give an example process flow. Requirement Management Team follows a Requirement Analysis process which is done for serving Requirement Derivation function. The output of Requirement Analysis process is Requirement Description Document. This document is used by Architecture Team in order to design architecture. The document is an input to Architecture Design process and this process creates Architecture Description Document which then be used in implementation process. Since there are two Sites taking role in development, Architecture Description Document is an input to two different development processes whose outputs are working software units. This process flow is shown in Process View in Figure 9. The Process View is instantiated from the Process Viewpoint definition of Table 3. We have not included some other details of this process flow such as the causal connections among the different steps.

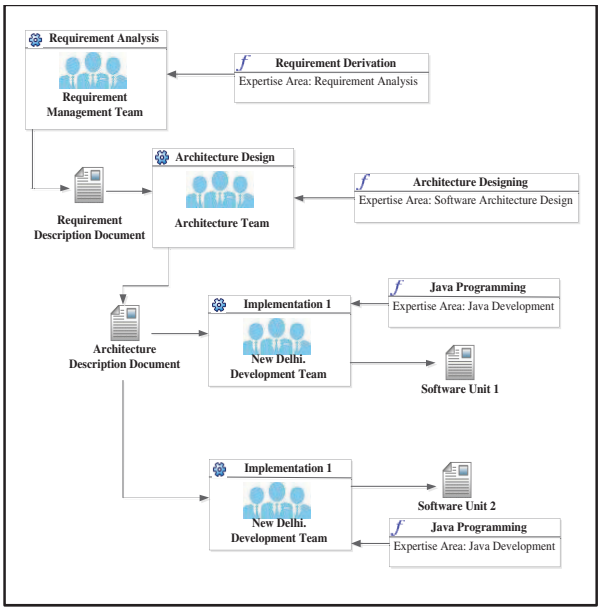

Figure 9. Example Case: Process View

The Migration view for the given case is given in Figure 10. As we stated before, migrations from site to site take place especially in the phases of the projects which require extensive collaboration and coordination. The Migration view illustrates the case in which a team in New York migrates to California. The travel is done in the context of Requirement Management and Architecture Design processes. Requirement Team also carries an external disk in order to carry some information that may be needed such as requirements documentations, some work templates, etc.

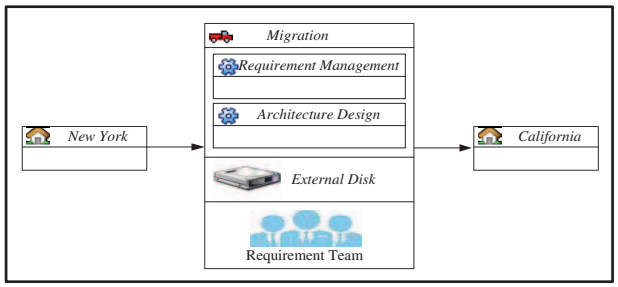

Figure 10. Example Case: Migration View

Figure 11 shows the Data view for the given example. The view shows the three sites New York, California, New Delhi and Sao Paulo which have their own local data storages. Further, each site has its own documents and local copies of other necessary documents kept in these local storages.

Due to lack of space we have only given four views. The designer can also use the other four viewpoints to define the corresponding views.

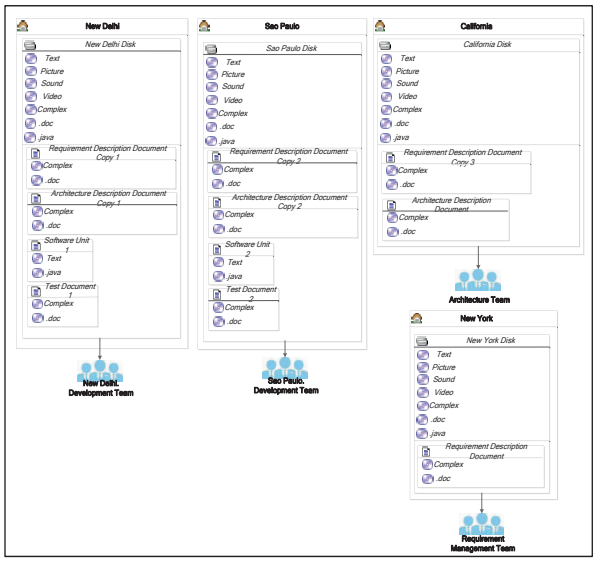

Figure 11. Example Case: Data View

V. RELATED WORK

Notably, architecting in GSD has not been widely addressed. The key research focus in the GSE community seems to have been in particular related to tackling the problems related to communication, coordination and control concerns.

Clerk et al. [6] 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.

Avritzer et al. [2] report on their experience in assessing the relationship between the dependency structure of a software architecture and the coordination needs among distributed development teams. They have used matrix models in the Global Studio Project Version 3.0, to represent both architectural dependencies and the coordination structure among the team members. The analysis of data gathered during the Global Studio Project Version 3.0 showed that design structure matrix (DSM) models representing the software architecture of GSD can help guide the task assignments in global software development projects.

Tool support has been named as one of the important challenges for GSD since it requires making software development tools and environments more collaborative [14]. Booch and Brown 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 [3]. A number of efforts have been carried out to support the idea of CDEs. Collab.net [7] is a commercial provider of CDEs, offering facilities for configuration management, bug tracking, task management and discussions. Spanjers et al. [15] discuss the system SoftFab, which automates the build and test processes in the context of multi-site projects. Caroll et al. [4] define the tool Jazz which supports rich synchronous communication, and promotes mutual awareness of coding activities within a development team. We believe that a common metamodel for GSD could help developing CDEs. The metamodel that we have provided in this paper could serve this purpose.

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. We have addressed these issues in our earlier work [16]. Here, we have proposed to apply a model-driven architecture design (MDA) approach to abstract away from platform concerns. Based on model transformation patterns as defined by MDA the portability and the interoperability concerns can be more easily addressed. The viewpoints in this paper could be uses as the source and target meta-models in the model transformation patterns.

VI. CONCLUSION

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. In this paper we have focused on the architecture design of GSD. Designing architecture for single systems is hard. Designing architecture for GSD is even more difficult due to the additional concerns for communication, coordination and control of distributed GSD teams. To support the architect in designing a proper GSD architecture we have provided a meta-model for GSD based on a thorough literature study. The meta-model captures the key concerns for designing GSD software architecture. We have identified the six different concerns including deployment, process, data, communication, tool and control. The meta-model served as a foundation for defining the architectural viewpoints for GSD. In our future work we plan to apply the approach in a real industrial setting. Also we aim to provide tool support dedicated towards modeling GSD architectures.

REFERENCES

[1] Agerfalk, P.J., B. Fitzgerald, H. Holmström Olsson, and E.O´ Conchuir. Benefits of Global Software Development: The Known and Unknown. in International Conference on Software Process, pp. 1-9, 2008.

[2] A. Avritzer, D. Paulish, Y. Cai. Coordination Implications of Software Architecture in a Global Software Development Project, Seventh Working IEEE/IFIP Conference on Software Architecture, pp. 107-116, 2008.

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

[4] M. Carroll and S. Sprenkle. Coven: Brewing Better Collaboration through Software Configuration Management. ACM SIGSOFT Foundations of Software Engineering, pp. 88-97, 2000.

[5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley, 2010.

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

[7] Collab.net http://www.collab.net, http://sourceforge.net, accessed June, 2011.

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

[9] C. Hofmeister, R. Nord, and D. Soni. Applied Software Architecture. Addison-Wesley, NJ, USA.

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

[11] P. Kruchten. The 4+1 View Model of Architecture. IEEE Software, 12(6):42–50, 1995.

[12] A.J. Lattanze. Architecting Software Intensive Systems: A Practitioner’s Guide, Auerbach Publications, 2009.

[13] R. Nord, P. Clements, D. Emery, R. Hilliard. A Structured Approach for Reviewing Architecture Documentation, Technical Note, CMU/SEI-2009-TN-0302009, SEI-CMU, 2009.

[14] B. Sengupta, S. Chandra, V. Sinha. A research agenda for distributed software development, In Proceedings of the 28th international conference on Software engineering, pp. 731-740, 2006.

[15] H. Spanjers, Ter, B. Graaf, M. Lormans, D. Bendas, R. V. Solingen. Tool Support for Distributed Software Engineering, in Proc of: International Conference on Global Software Engineering, 2006. ICGSE '06., pp. 187-198, 2006.

[16] B. Yildiz & B. Tekinerdogan. Meta-Model For Global Software Development to Support Portability and Interoperability, Sixth International Conference on Software Engineering Advances (ICSEA 2011), Barcelona, Spain, October, 2011.