The Data Interoperability Problem as an Exemplary Case Study in the Development of Software Collaboration Environments

Arturo J. Sánchez–Ruíz[1], Karthikeyan Umapathy[1], Jonathan Beckham[1], and Patrick Welsh[2]

[1] School of Computing
[2] Advanced Weather Information Systems Laboratory
University of North Florida,
1 UNF Drive,
Jacksonville, FL 32224
{asanchez,k.umapathy,jonathan.beckham,pwelsh}@unf.edu

Abstract. The Data Interoperability Problem appears in contexts where consumers need to peruse data owned by producers, and the syntax and/or semantics of such data—at both end points—are not already aligned. This is a very challenging problem whose instances can be found in practically every branch of human knowledge. In this paper we discuss the Data Interoperability Problem as an exemplary case study in the development of software collaboration environments. We define facets which prompt requirements that characterize the development of software systems as enablers of effective collaboration among data stakeholders, and also gauge the extent to which current technologies can be used to implement these software environments.

Keywords: software collaboration environments, aspects, variability axes, requirements, immersion, direct manipulation, second life.

1 Introduction

Consider the following fictitious problem inspired by a real–life scenario. Company 'X' manages human capital (i.e., workforce with different skill levels on a variety of fields) so it can offer services to many different clients on–demand. Each client has its own internal way of classifying human capital based on parameters such as field, education level, years of experience, and hourly rate, to mention just a few. How should 'X' consolidate all these different ways of representing what is essentially the same information across all its clients?

This is an instance of the Data Interoperability Problem, i.e., a collection of clients of 'X' produce data which 'X' needs to consume. However, such consumption cannot be seamlessly implemented because the data models used by 'X' to represent human capital information are not already aligned with those used by its clients.

E. Bertino and J.B.D. Joshi (Eds.): CollaborateCom 2008, LNICST 10, pp. 379–389, 2009.

In this paper we propose the Data Interoperability Problem can be considered as an exemplary case study in the development of software environments which enable data stakeholders to solve instances of this problem through collaboration.

Section 2 describes how we envision the collaboration among data stakeholders (i.e., producers and consumers) in order to characterize a model to be supported by software environments. Then, in Section 3, we identify facets associated with the development of software collaboration environments, namely: cross–cutting concerns, variability axes, and functional requirements.

In Section 4 we review various technologies which have the potential of either offering an exemplary perspective to the problem of developing these environments, or being used in their implementation. A comparison matrix, relating elicited requirements to technologies, is offered at the end of this section along with our comments on findings implied by this comparison.

The paper concludes with a discussion of additional related work (cf. Section 5), and our conclusions (cf. Section 6) which include a report of the current status of a project which has motivated us to participate in this workshop with the goal of interchanging points of view with other researchers in the area of collaboration.

2 Our Collaboration Metaphor

The following generic scenario is used to describe our Collaboration Metaphor (CM), i.e., our vision of how data consumers and producers interact with the goal of solving instances of the Data Interoperability Problem, via—what we refer to as—a 'just–in–time' approach.

CM1. Producers explain to consumers the nature of the data they own, including its syntactic structure and intended semantics. Visual representations of the producer's data are used to articulate explanations communicated to the consumer.

CM2. Producers and consumers engage in a question/answer dialog which ends when consumers express they have understood how to peruse the data.

CM3. Using the knowledge acquired from the previous step, consumers manipulate the data to satisfy their interoperability requirements.

This scenario is meant to be a general characterization of how humans approach the problem, not of how collaboration software is to behave. Our intention is to highlight the one–to–many relationship between the former (human approach) and the latter (software system) which is conceived of as an enabler tool. We do not mean to imply the described scenario shows the only—or more important—way humans approach the Data Interoperability Problem. However, we do claim this scenario is very relevant and common.

Consider the following real instance of the Data Interoperability Problem which deals with meteorological data, used to illustrate our metaphor. Given Global Positioning System Integrated Precipitable Water Vapor data (GPS–IPWV), which is typically produced in half–hour intervals, correct it for ambient

temperature and pressure in the vicinity of the master station for maximum accuracy. For **CM1**, the producer can explain to the consumer the structure and meaning of data stored (for instance) as a spreadsheet file. This can be accomplished by visualizing the data as a bidimensional array with appropriate labels for rows and columns. If the consumer is trained in meteorology, one would expect the loop in **CM2** to have a short duration. For **CM3**, the consumer uses his/her newly acquired understanding to determine what transformations must be applied to obtain the desired correction. At this point, the consumer might need the services of a software developer to actually implement the task at hand.

3 Facets Associated with the Development of Software Collaboration Environments

Using a problem/solution parlance, Section 2 characterizes the problem. In this section we offer various facets which characterize solutions.

3.1 Cross–Cutting Concerns

A design concern, i.e., an issue of high relevance in the context of the software system under development, is said to be cross–cutting if it affects several layers of this system. We are using the term 'cross–cutting' in the sense presented by Kiczales et alia [10], and the term 'concern' in the sense presented by Hilliard [19]. We have identified the following cross–cutting concerns (CC).

CC1. Target is End–User: This implies empowering users with software tools which allow them to effectively build solutions to their problems without demanding from them skills other than those naturally associated with their domain of expertise, and basic computing literacy. The September 2004 issue of Commnications of the ACM discusses various examples of this approach [1]. A related perspective is that associated with the area of Domain–Specific Software Development, which postulates end–users must be equipped with abstractions that are germane to their domain of application in such a way that solutions can be articulated—by the users—through a language which specifically targets this domain (i.e., a Domain–Specific Language) [3,7].

CC2. Variabilities are organically addressed by the software architecture: Variabilities are defined by axes on which a software system can evolve in the direction of being more generic. For instance, if the software system is a component which searches for an element in a collection, examples of variability axes include: the type of the elements, the data structure used to implement the collection, and properties of the underlying order relation used to determine whether the given element is in the collection or not. Examples of approaches whose raison d'être is to deal with variabilities include Software Product Lines [11,14,18], and Generic Programming [9,12,15]. We say variabilities are organically addressed by a software system if they have been modeled as part of the design of this software system. Examples of

practices which organically address variabilities include 'plugins', 'add–ins', 'skins', 'faces', et cetera. Section 3.2 mentions some variability axes for software collaboration environments in the context of the Data Interoperability Problem.

CC3. Visual Models can be Directly Manipulated: Data owned by producers and wanted by consumers can be explained in the context of a model, e.g., a data model. Consider, for instance, a relational model [5]. A visual model is a graphical representation of the data model which can be directly manipulated. In other words, the visual model supports interaction styles such as navigation, and selection/dragging of individual components/regions [4]. These interaction styles have semantics which are defined in terms of the semantics associated with the data model. For instance, in the case of a relational data model, the visual model can be based on the traditional relations–as–tables graphical representations, which would allow users to select cells (i.e., the semantics is a 'select' operation), and a region of the table (i.e., the semantics is a 'project' operation). It could be argued that **CC1** subsumes **CC3**. However, we think it is important to discuss it separately.

CC4. Resources are Automatically Generated: In the context of this paper the word 'Resources' denotes the results of the cooperation among stakeholders. Consider, for instance, the example presented at the end of Section 2. Instances of resources include: an actual software module which computes the appropriate correction transformations (e.g., written in the desired programming language), and maps showing the corrected data. The software module and the maps must be automatically generated by the collaboration environment from specifications that are—essentially—'gestured' by the user as s/he directly manipulates the visual models mentioned in **CC3**. In this case it could also be argued this concern is subsumed by **CC1**, and our reply to such argument is we believe it is important to discuss **CC4** separately.

CC5. Collaboration Decisions are Recorded: Strictly speaking, decisions that are made as part of the collaboration among stakeholders can also be considered as 'Resources', using the meaning discussed in **CC4**. However, we think it is important to differentiate between more 'tangible' resources, such as—for instance—Perl scripts [21], and 'knowledge–like' ones such as decisions made. If we consider again the example presented at the end of Section 2, we would like the collaboration environment to record the fact certain correcting transformations were used, so the environment can suggest them in the future if the same need arises. It could be argued this concern is subsumed by **CC4**, and therefore by **CC1**, but we believe it is important to discuss it separately.

CC6. Collaboration is by Immersion: Our working definition of 'Collaboration by Immersion' is the following. The software collaboration environment must make stakeholders feel as if they were together in a room equipped with various audio/visual tools which promote effective and efficient interactions among them (a.k.a. face–to–face environment). Additionally, our definition

requires cooperations enabled by software environments to be as fruitful, if not more so, than those counterparts enabled by face–to–face environments. Finally, our definition also requires for collaborations enabled by software environments to be considerably more efficient than those enabled by face–to–face environments. So far we have identified the following immersion styles: Virtual Reality, Second Life–like, and Elluminate–like. We have used actual commercial names to refer to the last two just because they can be considered as exemplars from the perspective of the concept of 'immersion style' we want to convey, not because we endorse these products. Virtual Reality is discussed in Section 5, while Second Life and Elluminate are discussed in Section 4.

3.2 Variability Axes

We have defined 'Variability Axes', in the context of cross–cutting concern **CC2** (cf. Section 3.1), as those on which a software system can evolve in the direction of being more generic. In this section we discuss the variability axes (VA) we have identified so far.

VA1. Domain of Expertise: Defined by the data model's structure (a.k.a. syntax), and its semantics which define the intended meaning of data and operations on them.

VA2. Type of Resource Generated: Examples of this type include programming language associated with the resource; the type of the resource (e.g., web page, programming module, 'face'[1], 'skin'[2], 'mashup'[3], et cetera); and knowledge (e.g., decisions made through cooperation).

VA3. Visual Models: The way data models are visually represented. For a fixed data model there can be many different ways to visualize it. As mentioned in the context of cross–cutting concern **CC3** (cf. Section 3.1), the visual model must support both the data model's syntax and semantics.

VA4. Type of Data Source: This type describes more 'concrete' representations of data models. Examples include: plaintext structured (e.g. XML[4], CSV[5]), plaintext unstructured, relational (e.g., MySQL, Oracle, MS SQL Server, PostgreSQL, et cetera[6]), and real–time streams of data.

3.3 Functional Requirements

Functional requirements describe relevant interaction scenarios between end–users and the software collaboration environment. We present them using a

[1] `http://developers.facebook.com/`
[2] `http://www.winamp.com/`
[3] `http://www.ibm.com/developerworks/library/x-mashups.html`
[4] eXtensible Markup Language. See `http://www.w3.org/XML/`
[5] Comma-Separated Values.
[6] To mention some well–known database management systems.

format akin to eXtreme Programming's (XP) 'User Stories' as discussed by Wake [23], and to 'Use Case Summaries' as discussed by Larman [6]. Quality Attributes (a.k.a. Non–Functional Requirements), although very important, are not discussed in this paper [16,17]. In the following list each requirement (R) includes, in brackets, the cross–cutting concerns with which it is associated (cf. Section 3.1).

R1. Users, easily, cost–free, and without the aid of a software developer or database administrator, import their data sources into the system [**CC1**, **CC2**, **CC6**].

R2. The system dynamically generates visual models from the imported data sources. The underlying software framework should also make it easy for developers to produce these dynamic models [**CC1**–**CC4**, **CC6**].

R3. The system allows stakeholders to interact, via direct manipulation, with the models (e.g., by clicking on the visual representations, by showing options, by dragging/selecting elements of the visual representation, by visually inspecting/traversing/navigating the models, et cetera) [**CC1**–**CC6**].

R4. The system supports a variety of data transformations and manipulations based on the imported data models (e.g., via a menu of options, and gestures such as point–and–click, select–and–drag, et cetera) [**CC1**–**CC6**].

R5. The system generates resources specified by user's actions [**CC1**–**CC6**].

R6. The system allows for multiple stakeholders to simultaneously collaborate and manipulate the models in real–time within an immersive environment [**CC1**–**CC6**].

4 Review of Some Technologies

Previous sections have characterized the task of building software collaborative environments based on our just–in–time collaboration metaphor (cf. Section 2) for the Data Interoperability Problem, according to the restrictions imposed by elicited requirements (cf. Section 3.3), which encompass the discussed cross–cutting concerns (cf. Section 3.1), and variability axes (cf. Section 3.2). In this section we attempt to gauge the extent to which reviewed technologies can be used to accomplish this task.

4.1 Technologies That Do Not Target End–Users Per Se

The following two technologies are worth mentioning even though they do not satisfy cross–cutting concern **CC1** (cf. Section 3.1), embeded in all elicited requirements. Altova's Mapforce[7] and Microsoft's SQL Server Integration (SSIS) Services[8] target software developers assisting them on several data mediation

[7] `http://www.altova.com/products/mapforce/data_mapping.html`
[8] `http://www.microsoft.com/sql/technologies/integration/default.mspx.`

and data mapping tasks approached in a non–collaborative way. Visual interfaces support the graphical specification of data operations with the corresponding automatic generation of implementation code in a variety of programming languages. These two products can therefore be considered as exemplary technologies which address cross–cutting concerns **CC2**–**CC4**.

Elluminate[9] is a technology that supports our collaboration metaphor (cf. Section 2) and characterizes one of the styles of collaboration by immersion we have identified. It can therefore be considered as an exemplar which aligns with our cross–cutting concern **CC6**.

4.2 Actual Implementation Technologies

Technologies presented in this section have been considered as candidate frameworks which can be used to implement software collaboration environments as per the elicited requirements.

Linden Research Second Life: Second Life[10] supports the construction of online virtual 3D worlds within which users can create and interact with objects, as well as among themselves via their avatars. Users create complex 3D representations from smaller objects (known as 'prims', or primitive objects) combining them to fit the required needs. Second Life possesses its own scripting language, called Linden Scripting Language, which can be used by world designers to add functionality to objects and to manipulate them within previously created inventories. One of the shortcomings associated with Second Life's computational model, however, is that in order to import externally fabricated 3D models into the users inventory, s/he must use the import method from within the environment which charges a monetary fee. Additionally, these imported models must then be manually placed in the environment and scripts have to be manually added to them, once they are imported, in order to provide functionality associated with direct manipulation. This dramatically limits the ability to dynamically create models driven by user–generated events as they manipulate constituting objects and cooperate among themselves. In conclusion, although Second Life's computational model supports collaboration and immersion, it (currently) neither supports the dynamic and event–driven creation of visual objects with which end–users can interact, nor it supports exporting generated resources.

Sun Microsystem Project Wonderland: Project Wonderland[11] is a framework for developing 3D virtual worlds which look similar to those developed in Second Life, although the former is currently not as advanced as the latter. Its open source environment supports multi–channel audio interactions allowing users to move amongst concurrent conversations in an attempt to simulate a real–life office. The framework comes with both server and client components which can be modified by developers to meet their needs. The

[9] `http://www.elluminate.com`
[10] `http://secondlife.com/`
[11] `https://lg3d-wonderland.dev.java.net/`

environment provides—much like Second Life—support for the creation of worlds in which users interact and collaborate among themselves in an immersive environment. Additionally, models created with software such as Blender[12] or Maya[13] can be imported into Project Wonderland environments via their World Builder tool. However, it currently does not support the creation of 3D models without the use of external tools. Project Wonderland seems to have the same limitations as Second Life when it comes to dynamically generating event–driven visual models.

ISO X3D: X3D[14] is an XML–based ISO standard which targets the representation of 3D graphic objects allowing users to programmatically create complex worlds. Additionally, it supports object extensions via an API, JavaScript, and Java. Although X3D supports the ability to program interactions with created objects, it does not provide tools that can be used to create a virtual collaboration environment. Further, importing data models would also require a custom solution, for instance written in Java, to interact with the X3D code.

Microsoft Silverlight: Silverlight[15] is a technology that allows users on the web to view high quality video and audio content created using the Microsofts Expressions Studio and the Visual Studio tool suite. The Expression Design tools allow designers to visually create graphics that are then expressed in XAML (Extensible Application Markup Language). XAML supports the generation of human–computer interface elements such as animations, and interactive controls which can be tied to .NET applications via the Windows Presentation Foundation (WPF) and cross–platform, cross–browser Silverlight Plug–ins. This allows developers to tap into the powerful .NET framework through supported programming languages (e.g., C#, Visual Basic, Iron Ruby, Iron Python, et cetera) to create complex applications with dynamic interfaces. The .NET framework provides a large amount of interfaces for various data sources.

Adobe Director: Director[16] is a multimedia authoring tool and plug–in engine that allows developers to create and publish interactive applications on the web, desktop, or to DVD. Commercial add–ins are available for Director to import data from various sources and the ability to export data is also available. Adobe products support SVG (Scalable Vector Graphics)[17], which is an XML–based model used to describe vector graphics.

4.3 Technology Comparison

Table 1 shows the reviewed technologies along with the elicited requirements **R1** to **R6**. A 'Y' in the matrix cell (T_i, R_j) means requirement R_j can be easily

[12] http://www.blender.org/
[13] http://usa.autodesk.com
[14] http://www.web3d.org/x3d/
[15] http://silverlight.net/
[16] http://www.adobe.com/products/director/
[17] http://www.w3.org/Graphics/SVG/

Table 1. Comparison Matrix

	Requirements					
Technology	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**
Second Life	N	N	Y	Y	N	Y
Project Wonderland	N	Y	N	Y	Y	Y
X3D	N	Y	Y	Y	Y	N
Sliverlight	Y	Y	Y	Y	Y	N
Director	Y	Y	Y	Y	Y	N

implemented with technology T_i. An 'N' means the requirement cannot be easily implemented with the corresponding technology.

4.4 Characterizing 'Killer Apps'

Clearly none of the reviewed technologies cover all elicited requirements. It therefore makes sense to envision applications that would result if one were to take the best of each of the reviewed tools. These kinds of systems are usually referred to as 'Killer Applications' or 'Killer Apps' for short. In this paper we offer two types of Killer Apps for the Data Interoperability Problem:

Killer App A: Collaboration by immersion is implemented à la Elluminate; direct manipulation and resource generation are implemented à la Mapforce/SSIS but targetting end–users and with the 3D–like quality delivered by Director/Silverlight/X3D/SVG. In this case, support for automatically recording knowledge product of the collaboration among stakeholders would need to be implemented on top of Elluminate, although stakeholders themselves can record this knowledge using ad–hoc methods.

Killer App B: Collaboration by immersion is implemented à la Second Life; direct manipulation and resource generation are implemented à la Mapforce/SSIS but targetting end–users and with the 3D–like quality delivered by Second Life augmented with Director/Silverlight/X3D/SVG. In this case, support for automatically recording knowledge product of the collaboration among stakeholders would need to be implemented on top of Second Life, although stakeholders themselves can record this knowledge using ad–hoc methods.

5 Other Related Work

Alternatives to our collaboration metaphor (cf. Section 2) are discussed in [8,22]. Virtual reality (VR) has been for many years the archetypical approach to immersion. We have not included this approach in our review because we consider the use of typically required 'tethers' impractical and invasive (e.g., googles, glasses, helmets, et cetera). Implementing immersion via 'organic user interfaces' seems like a fascinating idea worth exploring [20]. The collaboration environments for the Data Interoperability Problem we envision are framed by the

question "What is the power of computing, by machine and human together?" posed by Wing [13]. Another paper of ours discusses the elements of a generic and extensible architecture for software collaboration environments which target end–users [2].

6 Conclusions and Current Status

Building effective and efficient software collaboration environments is a very complex task. In this paper we have discussed the Data Interoperability Problem as an exemplary case study which can be used to determine the extent to which current technologies possess the level of sophistication required to implement these collaboration environments.

We have characterized a relevant collaboration metaphor and facets associated with the construction of software collaboration environments to support it, including cross–cutting concerns and variability axes. These concepts led to the definition of user–centric requirements which were used to determine whether reviewed technologies could support them. We concluded that none of the reviewed technologies were individually sufficient to implement a system which satifies the elicited requirements. Despite of this, we were able to envision two 'Killer Apps' as collages built from existing technologies.

We argue that, using a ranking of complexity which lists the most complex concern first and the least complex concern last, the complexity ranking of elicited concerns is **CC2**, **CC3**, **CC5**, **CC4**, **CC6**, and **CC1**. This ranking has suggested the project we are currently working on, which is the development of a software collaboration environment with the profile of **Killer App A** (cf. Section 4.4) for meteorological data.

References

1. Sutcliffe, A., Mehandjiev, N. (eds.): Special Issue on End–User Development: Tools that Empower Users to Create their own Software Solutions. Communications of the ACM 47(9) (September 2004)
2. Sánchez–Ruíz, A.J., Umapathy, K., Hayes, P.: Toward generic, immersive, and collaborative solutions to the data interoperability problem which target end–users. In: Proceedings of the 2^{nd} International workshop on Ontologies and Information Systems for the Semantic Web (ONISW 2008) (October 2008)
3. Sánchez–Ruíz, A.J., Saeki, M., Langlois, B., Paiano, R.: Domain–specific software development terminology: Do we all speak the same language? In: Proceedings of the seventh OOPSLA Workshop on Domain–Specific Modeling, ACM–SIGPLAN, New York (October 2007)
4. Shneiderman, B., Maes, P.: Direct manipulation vs. Interface agents. interactions IV(6) (November–December, 1997)
5. Date, C.J.: Database in Depth: Relational Theory for Practitioners. O'Reilly, Sebastopol (2005)
6. Larman, C.: Applying UML and Patterns: An Introduction to Object–Oriented Analysis and Design and Iterative Development, 3rd edn. Prentice–Hall (2005)

7. Domain Specific Development Forum, http://www.dsmforum.org/
8. Stahl, G.: Group Cognition – Computer Support for Building Collaborative Knowledge. MIT Press, Cambridge (2006)
9. Gibbons, J., Jeuring, J. (eds.): Generic Programming, IFIP TC2/WG2.1 Working Conference on Generic Programming, Dagstuhl, Germany, July 11-12, 2002. IFIP Conference Proceedings, vol. 243. Kluwer, Dordrecht (2003)
10. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., VideiraLopes, C., Loingtier, J.–M., Irwin, J.: Aspect–oriented programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
11. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern–Based Software Architectures. Addison Wesley, Reading (2004)
12. Jazayeri, M., Loos, R., Musser, D. (eds.): Generic Programming: International Seminar, Dagstuhl Castle, Germany. LNCS, vol. 1766. Springer, Heidelberg (2000) (selected papers)
13. Wing, J.M.: Five Deep Questions in Computing. Communications of the ACM 51(1) (January 2008)
14. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
15. Czarnecki, K., Eisenecker, U.W.: Generative Programming – Methods, Tools, and Applications. Addison Wesley, Reading (2000)
16. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison Wesley, Reading (2003)
17. Barbacci, M., Klein, M.H., Longstaff, T.A., Weinstock, C.B.:Quality Attributes. Technical Report CMU/SEI-95-TR-021, ESC-TR-95-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA (December 1995)
18. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison Wesley, Reading (2002)
19. Hilliard, R.: Aspects, concerns, subjects, views, ... In: Proceedings of the 1999 OOPSLA Workshop on Multi–Dimensional Separation of Concerns in Object–Oriented Systems (June 1999)
20. Vertegaal, R., Poupyrev, I. (eds.): Special Issue on Organic User Interfaces. Communications of the ACM 51(6) (June 2008)
21. Cozens, S.: Beginning perl, http://www.perl.org/books/beginning-perl/
22. Kaptelinin, V., Czerwinski, M. (eds.): Beyond the Desktop Metaphor – Designing Integrated Digital Work Ennvironments. MIT Press, Cambridge (2007)
23. Wake, W.: Extreme Programming Explored. Addison Wesley, Reading (2002)