MODEL-DRIVEN VIRTUALIZATION OF HEALTHCARE RECORDS

Daniel Alexander Ford
Department of Computer Science, IBM Almaden Research Center, 650 Harry Road, San Jose, CA, U.S.A.

Keywords: Model-Driven Development, Healthcare record management, Distributed Systems, UML, Eclipse Modeling Framework, EMF, Connected Data Objects, CDO.

Abstract: This paper describes an approach to solving scalability, distribution and accessibility problems associated with large collections of healthcare records. The technique described in the paper explains how to exploit Model-Driven Healthcare record implementations of the Health Level Seven (HL7) standards for Clinical Documents (CDA) and Continuity of Care Documents (CCD), such as those developed by the Model-Driven Healthcare Tools (MDHT) project to quickly and easily enable highly scalable, distributed healthcare applications. The MDHT project has developed models in the Unified Modeling Language (UML) of the HL7 standard that it uses to automatically generate high quality software implementations of the standard. The technique described in the paper shows how to retarget this code generation process to automatically create an alternative implementation using a technology called Connected Data Objects (CDO). This new implementation immediately supports scalable, distributed access to document collections. The paper then goes on to describe example applications made possible by the new capabilities provided by the alternative implementation.

1 INTRODUCTION

The Model-Driven Healthcare Tools (MDHT) project (Carlson, 2010) has the goal of producing open source software models and implementations of Health Level Seven (HL7) (Benson, 2009) standards for electronic healthcare documents such as of Clinical Document Architecture (CDA) and Continuity of Care Documents (CCD). The software modeling technologies used by the MDHT project are those provided by the Eclipse Foundation as part of its open source Eclipse Modeling Project. Notably, they include the UML2 project and the Eclipse Modeling Framework (EMF) (Steinberg, 2009).

The models created by the MDHT are represented in the Unified Modeling Language (UML) (Booch 2000) and can be automatically processed by the code generation facilities of the Eclipse Modeling Framework, to generate software implementations of the document standards in the Java™ programming language. The generated code reflects best practices for software implementation and is of a uniform high quality that is difficult to produce manually in any consistent manner. This Model-Driven approach to software development creates implementations with few, if any, errors, and which can be modified extensively and quickly by regenerating after making changes to the original model.

These generated implementations make the writing of other healthcare applications that use these types of documents much easier and much simpler. The MDHT project produces Java™ implementations of the various documents that can be directly manipulated by application code. The generated code also has extensive capabilities such as automatic XML serialization and deserialization, reflective object access and notification services, among many others. All of these features are directly available to application code.

A key point is that the MDHT project uses the EMF code generation capabilities to the maximum extent possible and consequently, the implementation contains no manually written code. This means that it can be easily reconfigured and automatically generated without the need to manually reproduce some portion of the implementation.

A general limitation of any Java™ application, generated, or not, is the finite amount of memory available to the Java™ virtual machine in which it is running. In the case of the MDHT project, this limits the size and the number of instances of healthcare documents that can be processed simultaneously in one Java™ virtual machine. Depending on the application, this may or may not be an issue, but it is a general problem that limits the range and scalability of potential applications.

This limitation also makes it more difficult for networked applications to cooperate and share data, as the *object graph* being manipulated on one Java™ virtual machine cannot extend to another Java™ virtual machine.

2 RETARGETING FOR SCALABILITY

Solving the scalability problem typically involves developing applications that make network access to a centralized repository, typically a relational database. Applications that take this approach tend to include implementation details that query and modify the database. These details take significant implementation and maintenance effort by skilled and experienced developers. Unfortunately, these efforts are usually orthogonal to the goals of the application itself.

The MDHT project is in the interesting position of having an implementation that is completely generated; this opens up the possibility of retargeting the code generating process to produce an alternative implementation that addresses the scalability. The Eclipse Modeling Framework contains a sub-project called Connected Data Objects (CDO) that enables exactly that alternative.

2.1 Connected Data Objects (CDO)

The CDO project supports the concept of a *virtual object graph*. This is a collection of Java™ objects that are persisted in a network accessible backing store, but which can be represented in multiple Java™ virtual machines simultaneously. The graph can be of an arbitrary number and size of objects. With CDO, the entire graph appears to exist simultaneously in the address space of any and all Java™ virtual machines that can make the appropriate network connection to the backing store. Objects in the graph can be manipulated with regular application logic with little reference to the existence of the backing store other than appropriate initialization and configuration.

CDO achieves this "magic" by configuring the EMF's code generation process to create Java™ classes that delegate all of their accesses (getters and setters) to reflective methods implemented by the internals of CDO. So, instead of generating "regular" Java™ classes that contain references to each other that are only valid within the context of a single Java™ virtual machine, alternative versions of the classes are generated which use abstract "CDO Object Id's" as references. Essentially, the CDO Object Id serves as a *virtual pointer* that can always be satisfied by the backing store. If a referenced class is not in the address space of the current Java™ virtual machine, CDO will retrieve it from the backing store. Thus, given that no particular generated class instance directly references any other generated class instance, they all can become candidates for regular Java™ garbage collection without the danger that the virtual graph maintained by the backing store will become disconnected or corrupted, this allows for arbitrarily large collections of objects.

For the MDHT project, retargeting its generated implementation to support a virtual object graph of clinical documents is as simple as changing three parameters controlling code generation. In fact, the CDO tool set in the Eclipse IDE includes a simple menu option to perform the necessary changes almost instantly; code regeneration itself, takes just a few seconds more to complete.

2.2 The CDO Server

The counterpart to the retargeted MDHT/CDO code running in an application is something called the CDO Server. Its job is to manage the backing store that persists the virtual object graph, and to provide network accessibility to the graph. The CDO Server can use a number of different databases to persist the virtual object graph, including MySQL, H2, and Objectivity.

A CDO Server can manage multiple virtual object graphs. It maintains a set of addressable "repositories," each of which can contain one or more EMF "Resources." A Resource is conceptually much like a file and is identified by a unique "path" that resembles a path in a hierarchical file system. A virtual object graph can be persisted across more than one Resource. There is a direct correspondence between a repository and a virtual object graph.

The CDO Server is very sophisticated. It can provide transactional semantics to maintain the integrity of a virtual object graph, and can provide historical "audit" views of the graph as it existed at any point in the past. It can also implement disconnected semantics in which two or more instances of a CDO Server can redundantly maintain a virtual object graph such that in the event of disruptions in communications, each can provide access to the graph. The use cases for this ability include local disconnected operation such as on a portable computer, or for geographically distributed servers that may experience communication problems. In either case, the CDO server will resolve differences between the two servers when communications are restored.

2.3 CDO Semantics and Capabilities

To facilitate multiple accesses to a virtual object graph, CDO provides transactional semantics and notification facilities. These allow different applications to modify documents in the graph and transactionally commit them to the CDO Server. Conflicting changes result in thrown exceptions for all but the first application to commit a change. Resolution of conflicting modifications is an application specific detail. In addition to transactional "write access," CDO also offers two other ways to access the contents of the virtual graph in an application. The first is a "read-only" representation of the graph that does not allow commits to the CDO Server. The other is an "audit" view that allows a read-only access to historic versions of the graph as it existed at specified times in the past.

The Eclipse Modeling Framework has extensive "notification" capabilities that allow application code to be notified of changes in instances of classes generated by the EMF. As part of the EMF, CDO also provides these capabilities, such that applications can be notified of changes to the object graph so that they can respond. For instance, an application might wish to be informed of the addition of new clinical documents to the graph so that it can process them and generate a report.

2.4 Advantages of Virtualization

There are a number of advantages of retargeting the MDHT implementation to leverage Connected Data Objects. The first is that little or no changes are necessary to application code written for the conventional or "legacy" (in "CDO-speak") implementation. The regenerated code retains the same Java™ interfaces as before, only the implementations of the interfaces are changed by CDO. This means that applications developed for the legacy version are often unaware that anything has changed and do not need to include complex code to manage a relationship with a relational database; they become "scalable" almost for "free."

The virtualization of MDHT also enables an entirely new set of distributed healthcare record processing applications and architectures. With a network accessible virtual graph of clinical documents, one can begin to develop applications that leverage this arrangement and use the graph as a common data structure on which to cooperate and coordinate their activities. For instance, some applications might be producers of clinical documents, while others might be consumers of some sort; some might be both by transforming or augmenting clinical documents for further processing by others. Since CDO supports multiple virtual graphs, it is easy to partition documents as necessary. The decoupling provided by CDO allows these types of applications to be developed independently and operate autonomously.

2.5 Virtualization Trade-offs

As with all things, there are trade-offs to leveraging a technology such as CDO to enable the virtualization of healthcare records. The first trade-off is performance. While, CDO is not necessarily slow and it does cache requested data, it still does need to resolve some object accesses via by connecting through a network to a CDO Server. This will always be slower than accessing clinical documents in a non-virtualized object graph. Further, there is extra complexity in reconfiguring the legacy MDHT model code and retargeting code generation for CDO. This process essentially creates two versions of the implementation, one legacy and one virtual; creating a software maintenance chore to keep them synchronized if both are desired. It is not difficult to do this, but it is a task that did not exist before.

The alternative approaches to creating a framework for a distributed scalable healthcare record infrastructure would seem to face similar issues, without necessarily having the advantage of automatic code generation. The transparency and lack of implementation required of an application developer using the generated MDHT/CDO implementation should drastically and dramatically shorten any development time when compared to alternatives that would require a developer to create, implement, debug, and maintain, their own distributed infrastructure. Trying to create an alternative the provided the capabilities offered by CDO would seem to require equivalent effort for a reimplementation of CDO.

3 EXAMPLE APPLICATONS

One of the first example implementations we experimented with was the distributed editing of clinical documents on multiple networked machines. The configuration was fairly simple and consisted of a CDO Server configured with an H2 database and a simple Eclipse based client that used the reflective EMF editor. This editor uses the metadata of the MDHT models (i.e., the names of the classes, etc.) which is embodied in special editor accessible classes created during code generation, to allow users to visually edit instances of a model, in our case, instances of Clinical Documents (CDA's).

Our initial tests uses another simple plain Java™ "loader" application that used the MDHT's generated XML deserialization abilities to load sample clinical documents stored on the local file system. These were then added to a CDO Resource maintained obtained from the CDO Server, and then committed.

With the Resource preloaded with example clinical documents, we then ran the client applications on three different networked computers, on which we opened identical CDO "transactional sessions" with the CDO Server. Within the context of the session we then were able to open the Resource in the editor and view the same collection of clinical documents on each of the three machines.

Selecting a portion of a document in the editor revealed the "properties" of that item in the Eclipse Properties View, where it could be modified. Saving the contents of the file triggered notifications that resulted in the changes being committed to the CDO Server. This caused the server to notify the other open sessions of the changes, triggering them to update their contents.

This simple example provided real-time distributed editing of clinical healthcare records with transactional semantics. The effort required to create the example was trivial in comparison to that required to manually produce such an implementation.

3.1 Distributed Healthcare Applications

The ability for an application to leverage the ability of the MDHT/CDO code base to access a virtual graph of interconnected healthcare records without significant implementation effort enables a new set of application domains that were previously difficult or too expensive to produce. It also significantly alters the architectures used to develop such applications as it becomes possible to independently implement and distribute various aspects of healthcare record processing by leveraging the loose coupling provided by the virtual object graph of healthcare documents.

One can envision network connected medical devices that operate as "first class citizens" in the healthcare record "universe." These devices would use MDHT/CDO to produce fully formed healthcare documents which would then be added to the appropriate CDO Server provided Resource which would persist them and notify others of their existence.

Extending this architectural concept further, we can imagine a suite of applications that "tap into" various virtual collections of healthcare records. Some, like medical devices, would be a source of such documents, while others such as nurse's stations or other user oriented applications would tend to be consumers. Other types of applications would monitor and transform, for instance, aggregating the outputs of all of the medical devices and logging their contents, creating summary reports for nurse's stations, or performing sophisticated semantic analysis to look for trends or anomalies.

The key point is that all of the devices and applications would use the same uniform representation of healthcare records and the same transparent technique to access their collections. Interchangeable interactions between existing and future applications would involve little difficulty.

4 CONCLUSIONS

The appearance of Model-Driven implementations of healthcare record standards, such as those produced by the MDHT project, has created new opportunities to easily solve scalability problems and to create new application architectures. The MDHT's use of the Eclipse Modeling Framework allows the generated implementation to be retargeted to use another EMF technology called Connected Data Objects. This technology allows the creation of virtual object graphs that allow applications to access and maintain collections of healthcare records of arbitrary size. These collections can be accessed, with transactional semantics, by multiple distributed applications simultaneously.

The effort required to achieve this ability is a trivial reconfiguration of the MDHT code generation parameters. This produces an alternative implementation of the HL7 standard documents that can be used in virtualized collections with no modifications.

The virtualization of healthcare records enables new types of applications and application architectures. These can be created with minimal effort in comparison to alternative manual implementations.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

REFERENCES

Carlson, D., https://mdht.projects.openhealthtools.org/

Benson, T., Principles of Health Interoperability HL7 and SNOMED (Health Informatics), Springer, 2009.

Steinberg D., Budinsky, F., Paternostro, M., Merks, E., 2009. EMF Eclipse Modeling FrameworkPaper templates. 2nd Edition. Addison Wesley.

Booch, G., Jacobson, I., Rumbaugh J., OMG Unified Model Specification, Version 1.3 First Edition: March 2000.