Model Driven Development of Inventory Tracking System*

Gan Deng, Tao Lu, Emre Turkay
Aniruddha Gokhale, Douglas Schmidt
ISIS, Vanderbilt University
Nashville, TN 37221

Andrey Nechypurenko
Siemens
Germany

Contact Author: Aniruddha Gokhale (a.gokhale@vanderbilt.edu)

**Work supported by grant from Siemens AG, Germany*

Keywords: Inventory Tracking System, Domain Specific Modeling, Model Driven Architecture, Model Integrated Computing, CoSMIC

An Inventory Tracking System (ITS) is a control and monitoring system that tracks the flow of goods and manages assets of warehouses. ITS is commonly used in warehouse management systems. The primary goal of an ITS is to provide convenient and reliable mechanisms to manage the movement and flow of inventory in a timely manner. An ITS should enable operators to configure warehouse storage organization criteria, maintain the set of goods known to the system, and track the inventory using GUI-based operator monitoring consoles.

The primary goal of the joint Siemens/ISIS ITS project is to apply the Object Management Group (OMG)'s Model Driven Architecture (MDA) [1, 2] together with component middleware [3] to configure various aspects of a warehouse management system.

The remainder of this document is organized as follows: Section 1 introduces the architecture of the ITS system, including the high level design of major components in the system; Section 2 discusses the advantages of choosing Microsoft Visio as our modeling tool, which is used to develop our warehouse modeling paradigm; Section 3 discusses our ongoing work on using OMG MDA in conjunction with middleware to facilitate the component synthesis; and finally Section 4 provides concluding remarks.

1. Architecture

The ITS architecture, illustrated in Figure 1, is designed in accordance with the CORBA Component Model (CCM) [3]. Each CORBA component representing different entities of the warehouse, such as cranes, forklifts and shelves, provides one or more *ports* that can be connected together with ports exported by other components. These ports include event sources and sinks, facets, receptacles, and attributes. The ITS implementation uses the Component Integrated ACE ORB (CIAO) [4, 6], which is a quality of service (QoS)-enabled CCM [5, 10] middleware implementation developed at Washington University, St. Louis and the Institute for Software Integrated Systems (ISIS) at Vanderbilt University, Nashville.

Figure 1 illustrates the key CCM components in the ITS architecture**:**

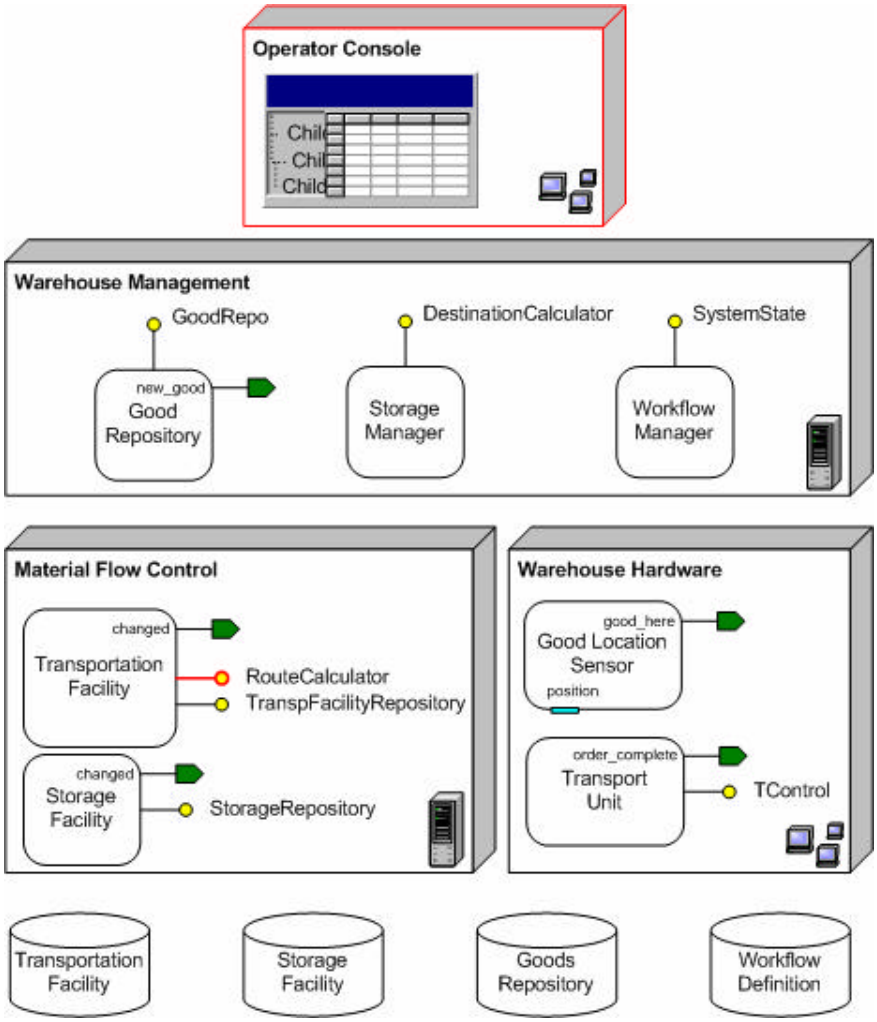

Figure 1: ITS Architecture

2. Model Driven ITS Development

The ITS project is developing and applying a set of modeling tools to automate the following three different aspects of ITS development:

1. Warehouse modeling that simplifies the warehouse configuration aspect of the ITS system according to the custom equipment available in certain warehouses, including roller conveyor and various types of cranes
2. Modeling of CCM container and ORB configuration aspects that are responsible for delivering the desired quality of service (QoS) properties to the ITS system
3. Modeling and synthesizing the assembly and deployment aspects of the components that implement the ITS functionality.

This paper focuses on the first aspect, which deals with the warehouse configuration.

Our ongoing work is addressing the remaining aspects of ITS development outlined above.

2.1 Warehouse Modeling

There are two main generic aspects in a warehouse model:

1. Transportation facility network, which includes information, such as the physical location and reachable areas, and properties, such as the toxicity of an item.
2. Appropriate available storage places, including their locations and properties.

The two aspects are blended together to give the model developer a convenient overview of the warehouse setup, which is similar to the architectural blue print of the warehouse. Mapping from the architectural blue print to the warehouse model should be intuitive to the domain expert, as well as to the model developer, so he/she can reuse the warehouse knowledge efficiently and conveniently

2.2 Choosing the Modeling Tool

After evaluating customer requirements, we have selected Microsoft® Visio® as our domain-specific modeling tool. Visio is a commercially supported graphic drawing tool with meta modeling capability, as well as the following desirable features:

2.2.1 Full Range of Technical Diagramming Capabilities.
In the drawing panel of Visio, numerous drawing related features are provided. For example, features, such as grid, docking point and object manipulation ability (resize, rotation, connection routing) are valuable for warehouse modeling. These intuitive features are important for domain experts who work on large-scale commercial ITS deployments.

2.2.2 Integrated Model Interpreter with Embedded Debugging Environment.
Unlike traditional tools that focus on a discrete segment of information, Microsoft Visio offers an integrated toolset for applications, development, and data modeling. Visio is shipped with an embedded Visual Basic® editor and debugging environment, which simplifies interpreter writing. C++/COM objects could be plugged in as well if desired.

2.2.3 Extensibility
Microsoft Visio provides support for database modeling, which includes complete database design, database schema, and Data Definition Language (DDL) script generation from conceptual and physical models. For example, in the warehouse configuration domain, we could connect the physical model with the associated database.

Moreover, Visio is shipped with many domain-specific paradigms (referred to as drawing types in Visio). Besides the major building blocks needed by the warehouse system, Visio also provides many other modeling paradigms, such as UML diagrams. The meta modeling ability

also makes Visio capable of extending the existing modeling paradigm to better suit the domain.

Figure 2 represents the Microsoft Visio screenshot where domain-specific model elements are available from the master panel (left side) and the right side contains the drawing representing the warehouse fragment comprising a moving belt, two cranes, storage rack and a forklift.

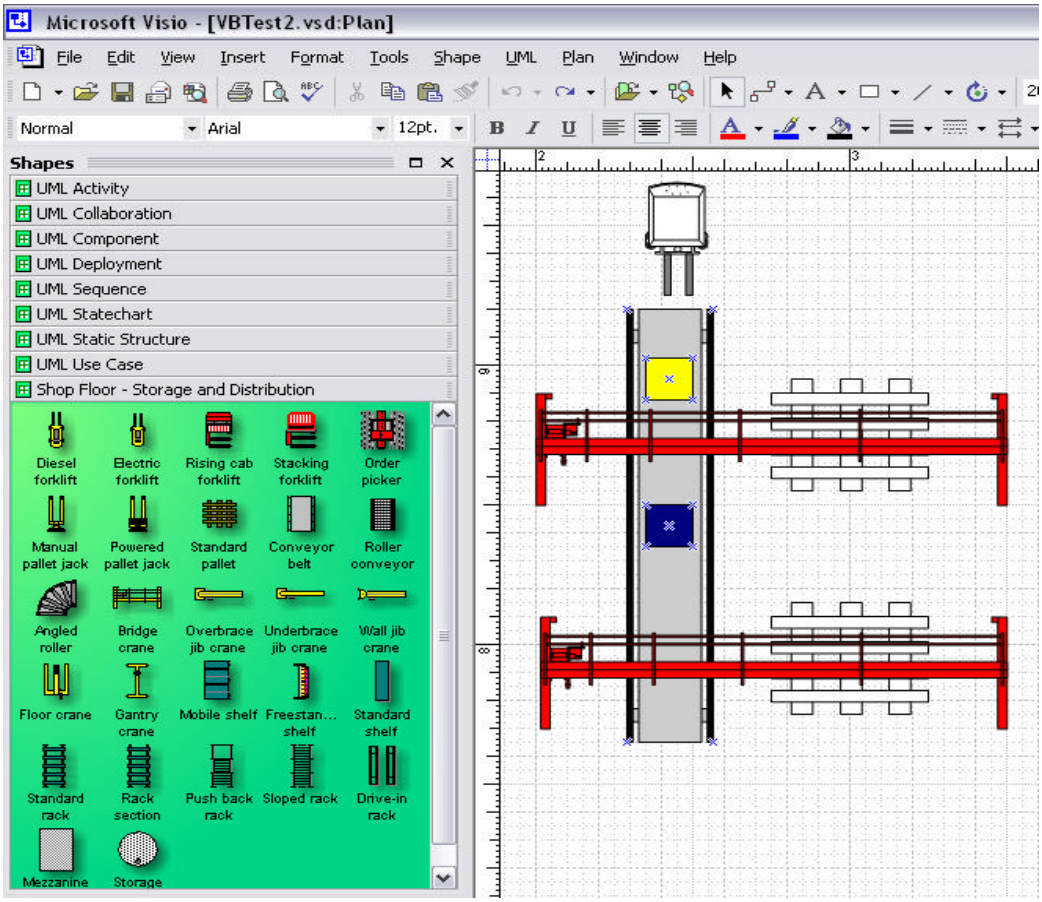

Figure 2: Microsoft Visio example

2.3 Model Interpreter

After creating the complete model for a desired warehouse configuration, the corresponding configuration artifacts are generated using the domain-specific model interpreter. The model interpreter we are developing for the ITS project is a Visual Basic based macro that be executed within Visio. In particular, the interpreter focuses on the following issues:

1. Certain location-related constraints can be checked inside the interpreter to validate the model, such as ensuring that the physical layout and configuration of the warehouse is legal and meaningful. For example, when a crane is on top of a storage place, we must make sure that the crane is capable of reaching all the storage cells of the place. Upon discovering potential conflicts, error or warning messages will be issued to the domain expert.
2. The underlying semantics of the graphic model can be abstracted from the model to

populate the databases of the warehouse system. The generated artifacts include the classes used to populate the databases and some the initialization process of the databases.

After running the model interpreter, the system is ready to start the component-based assembly and deployment process.

3. Ongoing work

One aspect of our ongoing work is the focus on building the run-time interactive model. For the current stage of the ITS development, we regard Visio as a static design tool. Thus, a model in Visio is transformed into a data model which will be used by the application. The Visio model itself will not be used after the static design phase. However, for the warehouse system, a dynamic provisioning tool is always desired. For example, with the dynamic provisioning tool, the operator of the system could observe the status of the transportation process being reflected on the GUI model, such as a blinking light indicating the failure of a transportation unit. This capability will allow the operator to dynamically reprovision the system.

To provide the features discussed above, we plan to make the domain specific building blocks executable. COM APIs will be used as the technology to implement this capability.

In addition to the Warehouse System Domain Specific Modeling Paradigm, we are also working on a domain-specific modeling tool suite, called Component Synthesis using Model Integrated Computing (CoSMIC) [7, 8], to model and synthesize the deployment and configuration of middleware components and the application's QoS requirements. In this regard the CoSMIC project is developing domain-specific tools for composing and deploying middleware-based applications. The CoSMIC tool suite is designed to model and analyze application functionality and QoS requirements and synthesize appropriate CCM-specific deployment metadata required to provision and enforce end-to-end QoS both statically and dynamically.

With respect to the ITS project, our goal is to integrate the Warehouse DSML and CoSMIC's DSMLS to address the full range of functionality comprising Warehouse Modeling, component-based distributed middleware configuration and the application assembly and deployment. By applying CoSMIC technology, we can take advantage of visual languages to compose components into component servers, which involves generating the directives to assemble semantically compatible application components from reuse repositories and determining the interconnections between these selected components. We can also use visual modeling languages to configure application component containers, which comprise generating QoS policies, such as threading policies or levels of security and fault tolerance, for the containers hosting the components. [9, 10].

4. Concluding Remarks

An Inventory Tracking System (ITS) is an example of a large-scale commercial system with multiple aspects of QoS requirements. This document describes how domain-specific modeling languages and model interpreters can simplify managing different aspects of ITS development. Our modeling tools are currently able to synthesize the database configuration and population aspects of ITS. Our ongoing work comprises using the OMG MDA technology in conjunction with the CORBA Component Model (CCM) for developing and deploying the components of the ITS. Modeling languages like UML and use case realization approaches manifested in our CoSMIC tool suite are used extensively for component development while other modeling paradigms are developed to handle the assembly and deployment aspects of ITS..

References

[1] Object Management Group, Model Driven Architecture (MDA), OMG Document ormsc/2001-07-01 edition, July 2001.

[2] Paul Allen, "Model Driven Architecture," Component Development Strategies, vol. 12, no. 1, Jan. 2002.

[3] Object Management Group, CORBA Components, OMG Document formal/2001-11-03 Edition (Jun. 2002).

[4] Nanbor Wang, Douglas C. Schmidt, Aniruddha Gokhale, Christopher D. Gill, Balachandran Natarajan, Craig Rodrigues, Joseph P. Loyall, and Richard E. Schantz, *"Total Quality of Service Provisioning in Middleware and Applications,"* vol. 26, No. 9-10, Jan 2003.

[5] BEA Systems, et al., *CORBA Component Model Joint Revised Submission*, Object Management Group, OMG Document orbos/99-07-01 edition, July 1999.

[6] Nanbor Wang, Krishnakumar Balasubramanian, and Chris Gill, "Towards a real-time corba component model," in *OMG Workshop On Embedded & Real-Time Distributed Object Systems*, Washington, D.C., July 2002, Object Management Group.

[7] Janos Sztipanovits and Gabor Karsai, "Model-Integrated Computing," IEEE Computer, vol. 30, no. 4, pp. 110–112, Apr. 1997.

[8] Aniruddha Gokhale, Balachandran Natarajan, Douglas C. Schmidt, Andrey Nechypurenko, Nanbor Wang, Jeff Gray, Sandeep Neema, Ted Bapty, and Jeff Parsons, *CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Component Middleware and Applications*, Proceedings of the OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture, Seattle, WA, November 2002.

[9] Douglas C. Schmidt and Fred Kuhns, *"An Overview of the Real-time CORBA Specification,"* IEEE Computer Magazine, Special Issue on Object-oriented Real-time Computing, vol. 33, No. 6, June 2000.

[10] Nanbor Wang, Krishnakumar Balasubramanian, and Chris Gill, " *Towards a real-time CORBA component model*," in OMG Workshop On Embedded & Real-Time Distributed Object Systems, Washington, D.C., July 2002, Object Management Group.