IMPLEMENTING A PROTOTYPE COMPUTER INTEGRATED CONSTRUCTION ENVIRONMENT

by

ANDREW L. GORLICK

BScEng, The University of New Brunswick, 1997

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Civil Engineering – Project & Construction Management Programme)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October 1999

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Civil Engineering

The University of British Columbia
Vancouver, Canada

Date 1999 October 7

Abstract

Construction projects rely on a large body of information produced by many sources at many levels of abstraction and detail. This is a major contributing factor to the fragmentation of construction data. One possible solution to this fragmentation is Computer Integrated Construction (CIC).

CIC provides methods for handling the information generated throughout the lifecycle of a project. The Construction Management Group at the University of British Columbia has developed a model for computer integration called Total Project Systems (TOPS). TOPS is a conceptual model with the potential to provide the basis for the development of a new class of CIC environments that are comprehensive, integrated, and flexible. Therefore, the main goal of this research was to develop a prototype CIC system based on the TOPS model and then to test the system to determine if the TOPS model provides a suitable basis for the development of the next generation of CIC systems.

The main approach was to develop a TOPS Implementation Prototype (TIP). This prototype was a multi-tiered architecture with repository technology at the core providing the persistence management functions. The TIP adopted an open, modular, and non-proprietary architecture so that any application could 'plug-in' to the system via standard TIP interfaces. The TIP also supported an architecture capable of being distributed via the Internet.

Repository technology was used to create a robust, integrated data source for the TIP. Repository technology is an innovative database technology developed by the software industry. It further enhanced the TIP architecture by making it extensible and by supporting data evolution and the

dynamic development of project schemas (i.e. schema evolution) without destroying information already stored in the database.

The TIP was successfully implemented and both a generic data browser application and a specific project management tool (a short cycle scheduling application) were constructed to test the potential of the prototype. The system performed well (given its prototype nature) and provided useful results on the TOPS approach for developing future integrated construction environments.

Table of Contents

List of Figures

Acknowledgements

The research represents the culmination of the past three years of my life. I am very proud of it but it would never have come to be were it not for certain people whose assistance, support, and encouragement were invaluable and must be acknowledged. My appreciation goes to: Dr. Thomas Froese – a most excellent supervisor; my parents Ruth and Lawrence – for their support and love; Dirk deRoos – the best friend I could ever ask for; Francesca Barth – the most wonderful woman to ever enter my life; and, my Lord and my God for His everlasting faithfulness. My sincerest thanks to each of you.

1. Introduction

1.1 Background on Computer Integrated Construction Environments

The essence of planning is to combine all aspects into one feasible process, which means that information has to be transmitted, transformed and combined. An increasing complexity of buildings and of the organisation of the construction process has made the transmission and sharing of information more difficult as there is a growing amount of information to be consolidated, distributed, and exchanged (Jägbeck 1998). The result is a reliance on a large body of information produced by many sources at many levels of abstraction and detail, which contributes to the fragmentation of the industry. This fragmentation, in turn, contributes to the poor record of overall productivity improvement in the industry. (Froese et. al. 1997)

One possible solution to this fragmentation is Computer Integrated Construction (CIC). CIC provides methods for handling the information generated throughout the life cycle of a project, promoting a more efficient and effective management process (Froese et. al. 1997). Computer integration, it is argued, will lead to improved communication among computer systems which will, in turn, lead to better information sharing among the project participants. Not only will it reduce errors and inefficiencies resulting from inaccurate, untimely, or missing information, but it will help foster better coordination and cooperation of the highly fragmented construction participants. (Russell and Froese 1997) Unfortunately, most of the software tools used to generate construction project information have not yet reached a state-of-practice where they interoperate effectively enough to support a data environment as complex and dynamic as a construction project. Organisations such as the International Alliance for Interoperability (IAI) are attempting to address this issue through their effort to standardise data, attempting to make

project information more explicit and self-contained (i.e. less context-dependent) (Jägbeck 1998).

However, one issue that still stands in the way of a successful implementation of CIC environments is the need for an underlying seamless, integrated data environment that supports this self-contained information. The Construction Management group at the University of British Columbia has been addressing the issues surrounding the development of a seamless integrated data environment to underlie the application environment as part of their Total Project Systems (TOPS) effort.

1.2 Background to Total Project Systems

TOPS is a conceptual model that provides a basis for the development of CIC environments. It defines a class of construction management computer systems that is defined by the following characteristics (taken from Froese et. al. 1997):

- *Comprehensiveness*: it includes a suite of applications that support a broad range of construction management functions.
- *Integration*: all applications contribute to and draw from a shared pool of project information.
- *Flexibility*: it operates in a highly modular, open, flexible, and distributed framework.

While each of these characteristics in itself is not revolutionary, when taken as a whole they describe a type of system that does not currently exist. TOPS is an attempt to push construction management (CM) computer tools past the point of 'critical mass,' where broadly-applicable computational models become the primary vehicle for practicing CM. This has the potential to significantly improve the manner in which construction projects are managed. (Froese et. al. 1997)

The ultimate goal of TOPS is to explore and demonstrate how integrated computer-based tools can increase the overall efficiency of construction management operations. It accomplishes this by supporting a broad range of functionality in support of traditional CM tools, such as scheduling, as well as emerging management tools, such as those that support construction methods selection or quality management activities. The key to efficiently supporting a broad and comprehensive range of functionality is to have a common source and structure of project information. Access and use of this integrated information must be available in an open-computing environment that is application independent. (Rankin et. al. 1999)

TOPS then, can best be thought of as a toolbox of construction management (CM) applications both contributing to and drawing from a common pool of seamlessly shared project information that is stored in a database(s) established on the structure of a core information model. (See Figure 1.1) Each tool requires specific information from the pool of project information in a variety of views and levels of abstraction. The system also handles the commitment of new contributions to the project information from each application. (Froese et. al. 1997)

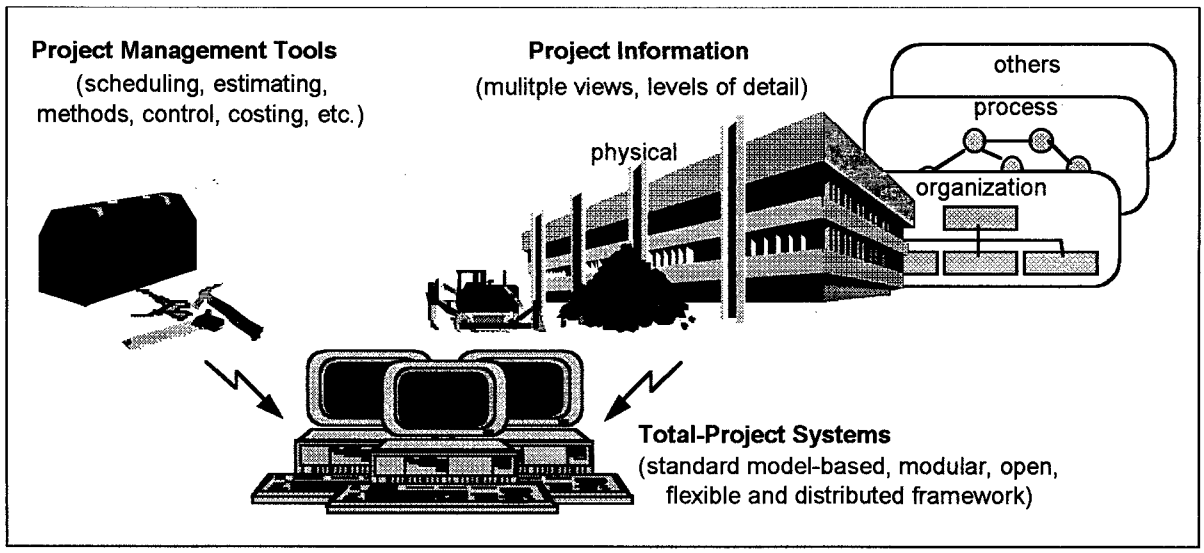

Figure 1.1 Visualising TOPS (after Froese et. al. 1997)

The TOPS model is founded on three principal research thrusts: application development, shared project representations, and system architectures and interfaces (Russell and Froese 1997). Much work has been done on application development (see Rankin et. al. 1998; Froese et. al. 1997; and Rankin et. al. 1997 for examples) and shared project representations (see Froese and Rankin 1998; Russell and Froese 1997; Yu et. al.1997; Froese 1996; and Fischer and Froese 1996 for examples). The focus of this thesis is the third research thrust, system architectures and interfaces.

1.3 Challenge and Objectives

Froese et. al. (1997) describe the architecture of TOPS as: *open,* in that it is not dependent on specific computing technologies (e.g., it is based on international data standards, it is platform independent, etc.); *modular,* such that a variety of specific applications developed by different sources can be brought to bear as appropriate to create the overall system; and *distributed,* by recognising the requirements of a variety of users and data sources. The challenge then, of this research is to develop and prototype a CIC environment/architecture that underlies and supports applications developed in this context. Named the TOPS implementation Prototype (TIP), this system will consist of a three-tier structure (separating the user interfaces layers, middleware (logic) layers, and data access layers from each other). Data will be structured on the International Alliance for Interoperability (IAI) data standards – Industry Foundation Classes (IFC) – and will be stored in a shared project information repository. The TIP will employ an open, modular, and non-proprietary architecture so that anyone that adopts a TIP standard interface can 'plug-in' to the system. The architecture will be extensible to support data evolution and the dynamic development of project schemas (i.e. schema evolution) without losing information in the database. Finally, the architecture will be distributed via the Internet. The goal of the TIP is to explore how the architecture of CIC systems should evolve in the future

to add project management functionality to the construction industry. The objectives of this research are:

1) to explore the implementation issues relating to TIP;
2) to describe the project environment that the TIP model will create in terms of process and data analysis and by using software diagramming techniques;
3) to conceptualise and implement a seamless, integrated data architecture based on a common product data model that consolidates project information;
4) to demonstrate the usefulness of such a data environment using two prototypes:
   a) the TIP data browser (see section 1.3.1); and,
   b) the Short Cycle Scheduling Application, SCSapp (see section 1.3.2).
5) to draw conclusions and recommendations.

1.3.1 TIP Data Browser

The TIP data browser is intended to play the role of an integrated project management system application that functions as a generic data control tool. It has been implemented in Visual Basic and Microsoft Repository. Similar in style to the Windows Explorer, it is designed to support the vast amounts of data generated on a construction project in an object-oriented, distributed, modular environment. It also demonstrates how a variety of project information can be retrieved and committed in a variety of views and levels of abstraction. Users are able to structure project models hierarchically by providing the system with information models (schemas) to underlie and structure project data. Microsoft Repository, an object-oriented metadata management facility, provides the capability to evolve data and to dynamically develop and modify project schemas without destroying information in the database. The system is modular in nature so that it can be supplemented with plug-in tools to accomplish a variety of project management tasks. The TIP data browser has the following additional sub-objectives specific to itself:

1) to use emerging repository technology to store project data;
2) to provide experience in implementing components in repository architecture and in implementing generic user interfaces;
3) to provide a consolidated view of all project information and to build new views and abstractions of project data;
4) to view and edit the schemas and models underlying the project information; and,
5) to add editing and long-term maintenance functionality to the project information base.

1.3.2 SCSapp

The primary purpose of SCSapp – the sample short cycle scheduling application – is to demonstrate how an integrated data environment functions inside the TIP architecture, how data is managed, and how such a system could be integrated into the construction industry's existing information technology infrastructure. A usage scenario will demonstrate the benefits to the construction process of such a data environment. SCSapp has been prototyped in Visual Basic and will be plugged into the TIP architecture to retrieve and commit project information to and from the shared pool of project data. It uses short cycle scheduling concepts developed by the Lean Construction Institute. SCSapp has the following additional sub-objectives:

1) to provide experience into designing and implementing specific project management applications and interfaces;
2) to provide experience in building multi-tier system architectures; and,
3) to assist project managers in developing short cycle schedules.

1.4 Scope

The TOPS idea presented in section 1.2 is a long-term research project by the UBC Construction Management Group to investigate an innovative approach to CIC. The TIP implements only a small portion of the TOPS approach – integrated architectures. Testing of the remainder of the TOPS approach, and the overall premises behind TOPS, is beyond the scope of this thesis. Similarly, any organisational or operational changes to the construction industry implied by the TOPS model also fall outside of the scope of this thesis.

The TIP is not intended to be production software. It is a proof of the conceptual integration models presented in TOPS. While the TIP does not have the full-fledged functionality of a production software system, it does provide the needed functionality to assist and further TOPS research and development efforts by the Construction Management group at the University of British Columbia. Adding new functionality to existing project management applications is also beyond the scope of the TIP.

That said, the development of the TIP is divided into several major components. The first component is the design and development of the TIP architecture. The architecture has been developed and constructed using conventional software modelling tools such as data flow diagrams and the Unified Modelling Language (UML).

The second component is the development of the data environment. The data environment consists of a data storage tool and a data browser. Microsoft's Repository technology has been used as the project data repository. While other commercial repositories are available and an entirely new repository could be developed, these other options have not been considered. The data browser has been coded in Visual Basic and structured in an object-oriented manner. The

International Alliance for Interoperability's Industry Foundation Classes were selected as the model upon which data stored in the repository is structured. Other data structures are available, however, the International Alliance for Interoperability appears to be the industry favourite at the present time and is the standard adopted in this research.

The third component is the development of the sample application to demonstrate how applications will interact with the TIP data environment. A short cycle scheduling application was selected because there is a need for a variety of information including: work task information, resource information, and scheduling information. It is felt by the author that a short cycle scheduling application makes sufficient use of the IAI data schemas when compared to more traditional and monolithic applications such as an estimating or scheduling package. The sample application has been coded in Visual Basic and structured in an object-oriented manner for the same reasons as the second component. The short cycle scheduling functionality was based on the concepts put forward by the Lean Construction Institute.

1.5 Reader's Guide

Chapter 1 of this Thesis covers the introduction to the subject matter and the research objectives. Chapter 2 is an explanation of the point of departure of this research and the contributions made to the state of knowledge. Chapter 3 describes the conceptualisation and development of the TIP prototype. Chapter 4 describes the implementation of a sample short cycle scheduling application that is used to test the TIP prototype. Chapter 5 presents a usage scenario and discusses the implications of the TIP. Chapter 6 presents the results and recommendations borne out of this research. Appendix A is a brief example of some programming code taken from the TIP prototype.

2. Point of Departure

2.1 Current State-of-Practice CIC Systems

This chapter presents some of the noteworthy research efforts that are representative of the current state-of-research into CIC systems.

2.1.1 ICIM

ICIM, the Integrated Construction Information Model, is an effort of the US Army Corp of Engineers Research Laboratories (USACERL) to develop an object-oriented information model that integrates product and process information to support communication and collaboration over the life cycle of a project (Stumpf et. al. 1996). ICIM is an object model used to model construction management processes. It is a static model that does not support extensibility. The ICIM object model forms the basis of the Agent Collaborative Environment (ACE). ACE functions as a middleware layer for the rest of the system, allowing components in the system to communicate with each other so long as they share the ICIM object model. The research efforts have focussed on two main technical areas: agent-based design tools and collaboration among agents in heterogeneous systems. Two agents have been developed: the Construction Planning Agent (CPA) and the Construction Project Control Agent (CPCA). The objective of the CPA is to support generation of a preliminary cost estimate and schedule for facility designs to assist the facility designers in evaluating alternative designs. The CPA integrates information from Microsoft Project, Excel, and AutoCAD. The CPCA was developed to manage progress information, update activities in the schedule, and maintain a record of construction. It is implemented in C++ and ObjectStore and persists (or provides long term storage of) the information generated by the CPA.

2.1.2 PerDiS

PerDiS (persistent, distributed, and shared memory) is an attempt by Sandakly et. al (1998) to build virtual enterprises to support concurrent engineering in Europe. Memory is shared between all applications, even between those located at different sites or running at different times. This shared memory is the shared store of the virtual enterprise. Essentially, data is continually cached and, whenever that data is modified, notifications are sent to pre-specified users explaining that a given set of data has been changed. PerDiS has a two-tier architecture containing two types of process: application processes and the PerDiS daemon. Applications use interfaces to connect to the PerDiS daemon which may then retrieve or commit information from other applications.

2.1.3 CORCE

CORCE (Consolidated Object Repository for Civil Engineering) is an object-oriented database developed by deMonsabert and Lemmer (1997) that is used to store and maintain civil engineering object/class hierarchies that comprise civil engineering software applications. The purpose of CORSE is to improve the efficiency in civil engineering applications development from the reuse of objects and classes. The database stores object design information for civil engineering applications. The CORCE database contains a list of class attributes and methods as well as inheritance, polymorphism, and encapsulation feature. It also contains general information regarding existing software to encourage the reuse of class definitions and objects among applications. The CORSE database also incorporates application information for both object-oriented and structured systems. This permits inquiries from developers regarding the object-oriented nature of systems. Similarly, information from structured applications can be used by an object-designer when constructing class methods.

2.1.4 Condor

Condor is an European effort to electronically manage the documents on a construction project through a model-based approach to information representation and structuring. The purpose of Condor is to address the following questions: is it possible to have, at any time, knowledge of the existence of a potentially useful piece of information used in a given project? Is there any easy mechanism to access transparently this information, update it (if it is inconsistent, and cross-reference it to another piece of information), while keeping the entire (fragmented and distributed) project information base consistent? The architecture developed to address these questions consists of integration services (implemented as class libraries), application interfaces which define the interfaces to the integration services, and adaptors which provide the mapping between the application interfaces and each of the documents to be integrated. (Rezgui et. al. 1999)

2.2 Author's Contribution

The author's principal contribution is the design and implementation of the TOPS Implementation Prototype (TIP). The TIP architecture differs from the current state-of-practice CIC architectures in the following ways: project information consolidation, middleware, repository technology, and schema extensibility. Each of these will be discussed in turn.

2.2.1 Project Information Consolidation

The TIP has a consolidated information base that has self-contained meaning (i.e. the information does not need to be exposed through an application to have meaning). By consolidating the project information, the TIP architecture creates a root data source that embodies the sum total of the project knowledge (and may capture information beyond what is exposed by standard applications). In the TIP environment, the root source can be interacted with

directly through the TIP Data Browser to build new views and abstractions of project data to create much more robust and powerful representations of project information. The author proposes that an architecture which supports a root source, and operations on its contents, has far greater potential to support a CIC environment than the potential offered by individual applications operating on discrete sets of data.

2.2.2 Middleware

The TIP architecture differs from state-of-practice CIC environments in that it is modular and distributed. The modular nature of the TIP allows virtually any CM application to be attached to the system. Given that applications will interact with the TIP through standard interfaces, the applications can be updated at any time without affecting the integrity of the remainder of the system. Part of the standard TIP application interface is the ability to convert data to the Extensible Markup Language (XML) before transmitting data between other applications or layers in the TIP architecture. XML is an application of the Standard General Markup Language (SGML) (as is the markup language HTML). It is a standard Internet protocol that is rapidly gaining support around the world as a standard of distributed document communication (Mace et. al. 1998). Converting the TIP data to XML strings allows the data to be served over the Internet in a distributed system.

This modular and distributed nature is due to the author's introduction of the Lightweight Data Access Interface (LDAI). The LDAI is the data access, or "middleware", component that defines the functional interface between an application and other applications or between an application and one or more collections of data. Middleware provides an isolation layer of software that shields application developers from the 'plumbing' by presenting its own enabling layer of application interfaces. This layer hides the differences incurred by a heterogeneous environment.

In effect, such a layer decouples otherwise coupled applications from any dependencies on platform specific application interfaces. (International Systems Group 1997)

2.2.3 Repository Technology and Schema Extensibility

Repositories are a relatively new class of software based on object-oriented database technology. They are access and life cycle management systems for metadata. Developed by the software industry to manage and support the software development process, repositories act as integrated management platforms for metadata, providing storage, consolidation, and versioning services as well as acting as the hub for data and component definitions, development and deployment models, reusable software components, and data warehouse descriptions. Metadata is descriptive information about the structure and meaning of data and about the applications manipulating data. (Microsoft 1999a) Integrated metadata management provides an enterprise with the ability to define a global and consolidated view of the structure and meaning of its applications and data – information that is usually scattered throughout the enterprise and buried in individual files, catalogues, or databases. (See Figure 2.1) This would facilitate the sharing and reuse of descriptive information between applications, and support version management and tool interoperability. (Microsoft 1999b)

Repository technology enables the extensibility of the TIP architecture. Extensibility is the ability to modify the models and schemas underlying project information stored in the repository without destroying that information. Clearly, information models will change over time and, therefore, it is important to allow the underlying conceptual model used by the project actors to be altered and to evolve over time – without affecting the overall consistency of the project information base. (Rezgui et. al. 1998) Repositories accomplish this by storing information

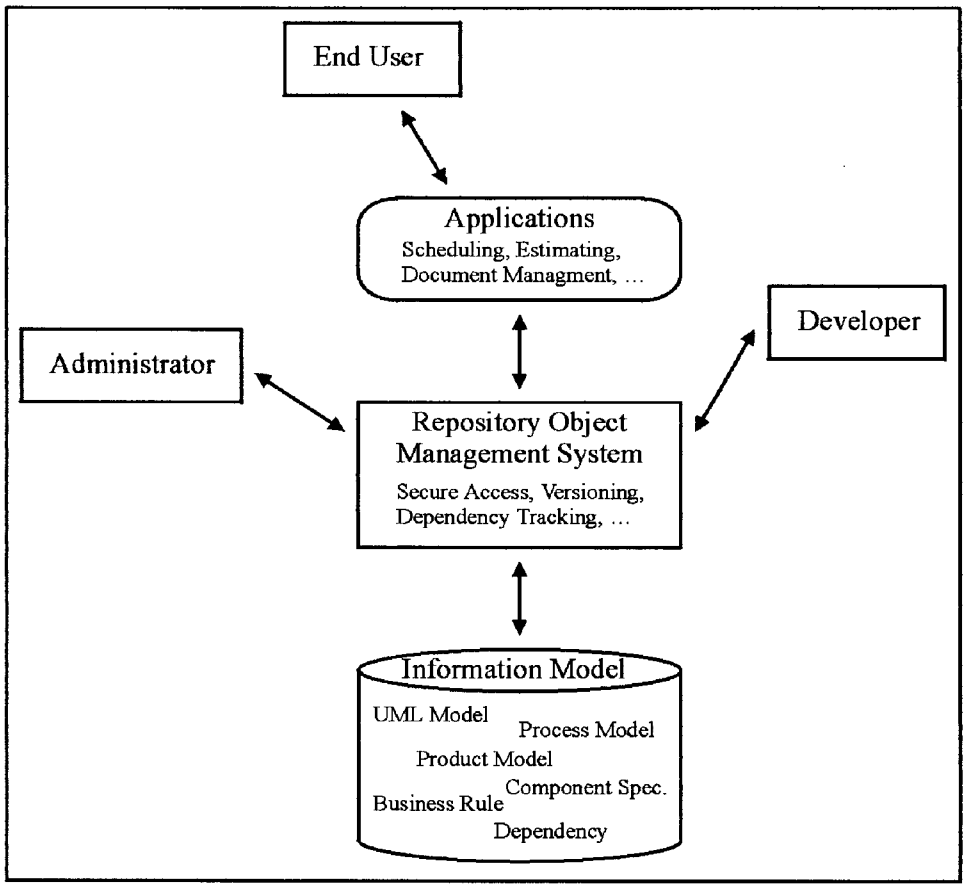

Figure 2.1 Repository as a hub for information (after Microsoft 1999b)

models together with the information they describe making the data self-descriptive. A tool is therefore able to query the contents of an information model and use the results to adapt its processing. This makes it possible to evolve data in response to information model changes, since tools depend only on the information model, not on the stored representation of data. This allows the dynamic development of the project model without destroying the information in the database. (Microsoft 1999a)

The author has adopted Microsoft's repository technology as the persistence manager for the TIP data environment. In this decision to adopt Repository, an important consideration was made regarding the type of database needed to support the TIP architecture. Essentially, there are three main alternatives for providing persistence management services: file-based data storage,

relational database management systems, or object-oriented database systems. File-based systems are quite simple but inefficient when dealing with large data sets. Relational databases have the advantage of being the most commonly used and available database system (e.g., Microsoft Access). Object-oriented database systems fit well with the type of object modelling being attempted in TIP but they are still relatively uncommon when compared to relational systems. The TIP architecture required a persistence manager that was object-oriented yet common enough to be easily integrated into existing software architectures. Microsoft Repository was an excellent response to these requirements. It adds a true object-oriented database layer on top of a relational. Furthermore, because it is a Microsoft tool, it nicely with the other Microsoft tools development tools used for this project.

3. Prototype Development

The basic idea behind Computer Integrated Construction (CIC) is that users working with one application can exchange data with users of another application through an interface to a custom data exchange mechanism. (Froese 1999) Figure 3.1 illustrates this.

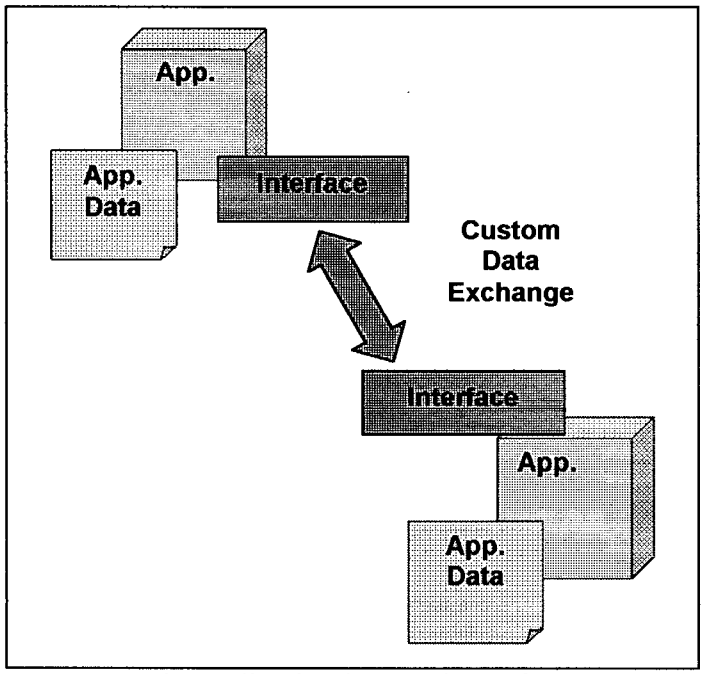

Figure 3.1 Basic application integration (after Froese 1999)

However, such a simple model of CIC leaves many issues unaddressed. In a multi-application environment that adopts this model of CIC, any single source application requires that an interface be developed for every target application with which it interacts. Different target applications will have different functions and may require a unique data set from the source application. As a result, there is no guarantee that components for one interface can be reused by another interface as the target application may require the interface to adopt an architecture unique to its own data needs.

This type of solution is increasing in popularity with the advent of office suites of applications such as Microsoft Office or WordPerfect Suite. Unfortunately, it is an inadequate solution for the

construction industry, in large part, because the industry makes use of more than just office suites. Applications such as AutoCAD, Primavera, Prolog Manager, etc. all have their own proprietary data formats with their own unique object models underlying the data. A large CIC system may need to consist of many of these applications. Building the interfaces (as well as maintaining and upgrading those interfaces) between each of these products would be time consuming and expensive.

In comparison, the basic idea behind TOPS is to develop application interfaces that map the data from each application to a common data model. This interface can accommodate data exchange with a central repository and can communicate with any other application that has an interface that also maps data to the common data model. This architecture is illustrated in Figure 3.2. More complex than the architecture in Figure 3.1, the TOPS architecture includes a common data model, databases, applications, user interfaces, and an object browser. The object browser is not part of any particular application but is required in order to view and manipulate all project information (Froese et. al. 1997).

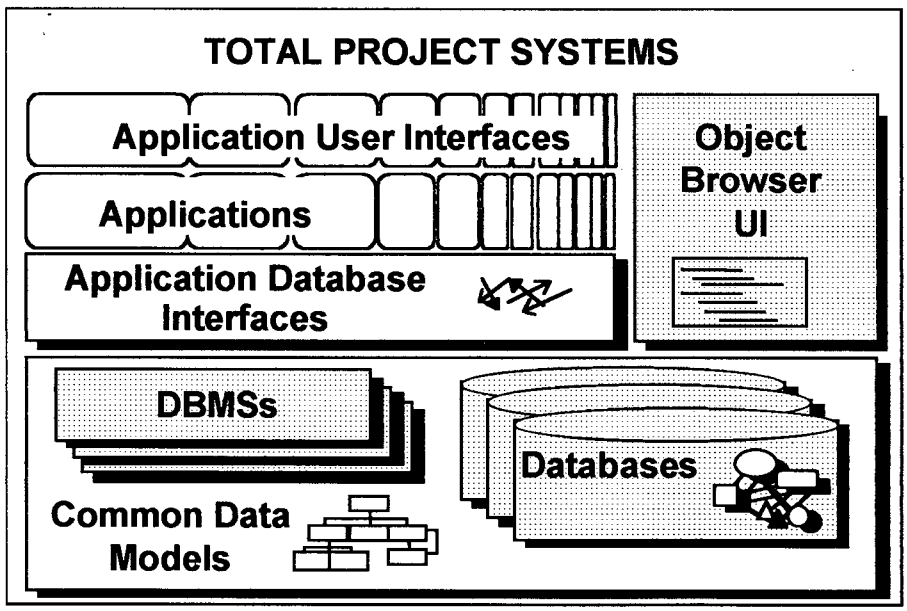

Figure 3.2 Components of TOPS (after Froese et. al. 1997)

All applications both contribute to and draw from a shared pool of project information that is stored in a database(s) established on the structure of a core information model. Application modules are connected to the system using standard interfaces. Thus, they can be updated without the need for changes to other parts of the system. Not only is project data shared, the application modules are also shared by different participants at different locations. As the applications and project data are transparent, users must simply send requests to the system through the user interface, invoke appropriate modules, and input the appropriate data. (Froese et. al. 1997)

TOPS only requires a single standardised interface to be written for each application. This interface references application data to a common product data model before persisting the data in a shared repository where any other application, regardless of function, may access that set of data. The accessing application's interface then converts the data back into a proprietary format (if necessary) useable by the application. Because applications are separated from data by a layer of logic, they may be upgraded or otherwise changed without affecting data access or the integrity of the data. Further, the interface may contain logic that supports emerging Internet protocols (such as XML) making the overall architectural framework highly distributed. Clearly, the TOPS model has the power to form the core of future CIC development efforts if it can be effectively implemented.

3.1 TOPS Implementation Prototype Architecture

The TOPS Implementation Prototype (TIP) architecture is presented in Figure 3.3. Except for the introduction of the LDAI (Lightweight Data Access Interface) and Microsoft Repository, the architecture is consistent with the architecture proposed by TOPS (see Figure 3.2). Microsoft's

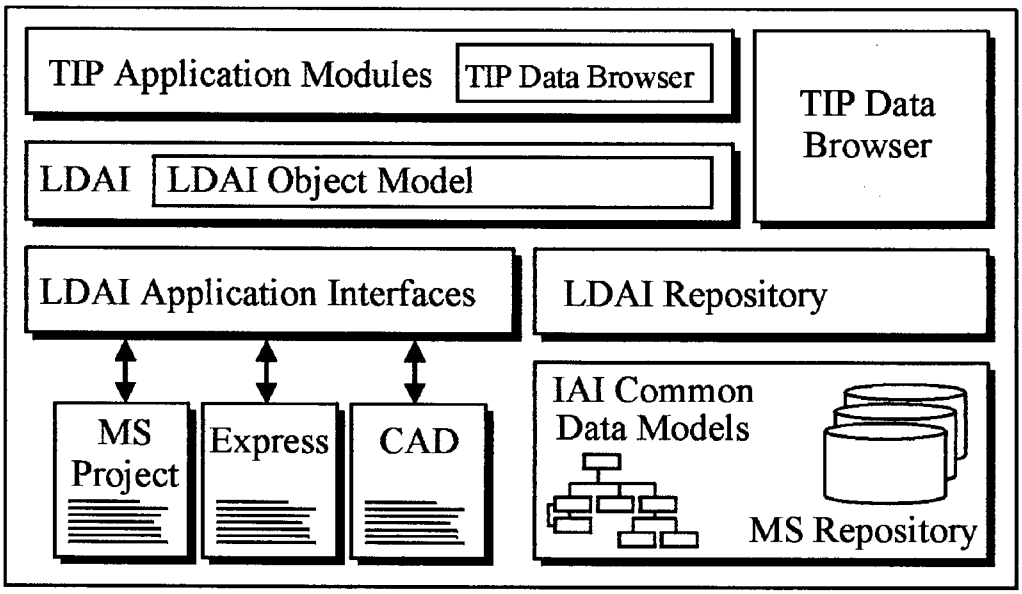

Figure 3.3 TIP Architecture

repository technology is used to persist the project information. It is exposed to the rest of the system through the LDAI Repository layer. This layer contains the repository engine and object model. Any part of the system that attempts to communicate with the repository must do so through this layer. Schemas are stored in the repository as IAI models with possible extensions for application-specific functionality. An OLE DB server is used to serve the repository, making it accessible to the Internet. The LDAI is the middleware component that defines the functional interfaces between an application and one or more collections of data. Although only three backends are shown in Figure 3.3 (MS Project, Express, and CAD), the TIP system can implement the LDAI with any number of backends to support the interoperability needs of any enterprise.

At the application level, two classifications of applications exist. The TIP data browser is a general-purpose user interface to project data and is an integral component of the TIP. This data browser can serve as a separate tool or application, similar to that of the Windows Explorer tool in Microsoft Windows. Its purpose is to allow users to enter, view, and manipulate project data

as well as view and modify the underlying models and schemas. The data browser interface can also be incorporated into other, more specialised applications (as shown by the combined TIP Application/Data Browser element in Figure 3.3). The TIP data browser can operate on the repository directly or through the LDAI. Finally, the TIP Application represents any standalone module that plugs into the LDAI. Any number of construction management applications may be developed – estimating, scheduling, quality control, or document management systems for example – so long as the required underlying schemas have been installed to support the application module.

All communications occurring between modules or between layers are standardised as an XML string. XML is a standard Internet protocol that is rapidly gaining support around the world as a standard of distributed document communication (Mace et. al. 1998). Standardising internal communications and data flow as XML strings supports a distributed architectural framework with the Internet as the controlling medium. Implementing the data flow in the TIP architecture as XML strings realises an additional, secondary benefit: data becomes readily accessible to users outside of the TIP. Since web browsers such as Microsoft's Internet Explorer and Netscape's Navigator are XML-capable, they are able to read and display XML data to users as a formatted XML document. Therefore, even users who do not have access to the TIP will be able to view any, and all, data stored in the project repository.

3.1.1 The LDAI

Applications interact with the integrated system through data access interfaces that are structured according to underlying schemas or data models (in this case, the LDAI object model) that are in turn structured on underlying assumptions about the structure of primitive concepts such as objects, classes, etc. (i.e., the metamodel). The LDAI (lightweight data access interface) defines

a functional interface between an application and one or more collections of data. The LDAI is a middleware component similar to other middleware standards such as the ISO STEP Standard Data Access Interface, SDAI (see ISO (1995) for an explanation of the SDAI). The LDAI was developed rather than directly adopting the SDAI because the TIP's requirements for a middleware layer are quite different since the TIP is not production software whereas the SDAI is more intended to support production software. The TIP is designed to support rapid prototype development in an area where "standards" are frequently changing. The LDAI has the following defining characteristics:

- It is very simple to work with. It is implemented in Visual Basic and it provides only the most essential and generic data access functionality (it can be thought of as a stripped-down version of the SDAI).
- It can be used as a "front end" interface for a wide variety of data access "back ends", such as EXPRESS, STEP Part 21, or XML files, or SDAI or Microsoft ADO data access interfaces (and it can act as a translator between these different formats).
- Only the smallest possible set of core metamodel data structures are statically encoded in the LDAI; other aspects of the metamodel and all project data models are dynamically loaded at run-time (i.e. late binding) and these models can be altered at any time.

The LDAI object model is presented in UML notation in Figure 3.4. Each object in the figure is a class. Classes may contain attributes and methods. Attributes are specified above the methods in an individual class diagram. Associations represent relationships between instances of classes. An association that has an open arrow on one end indicates inheritance.

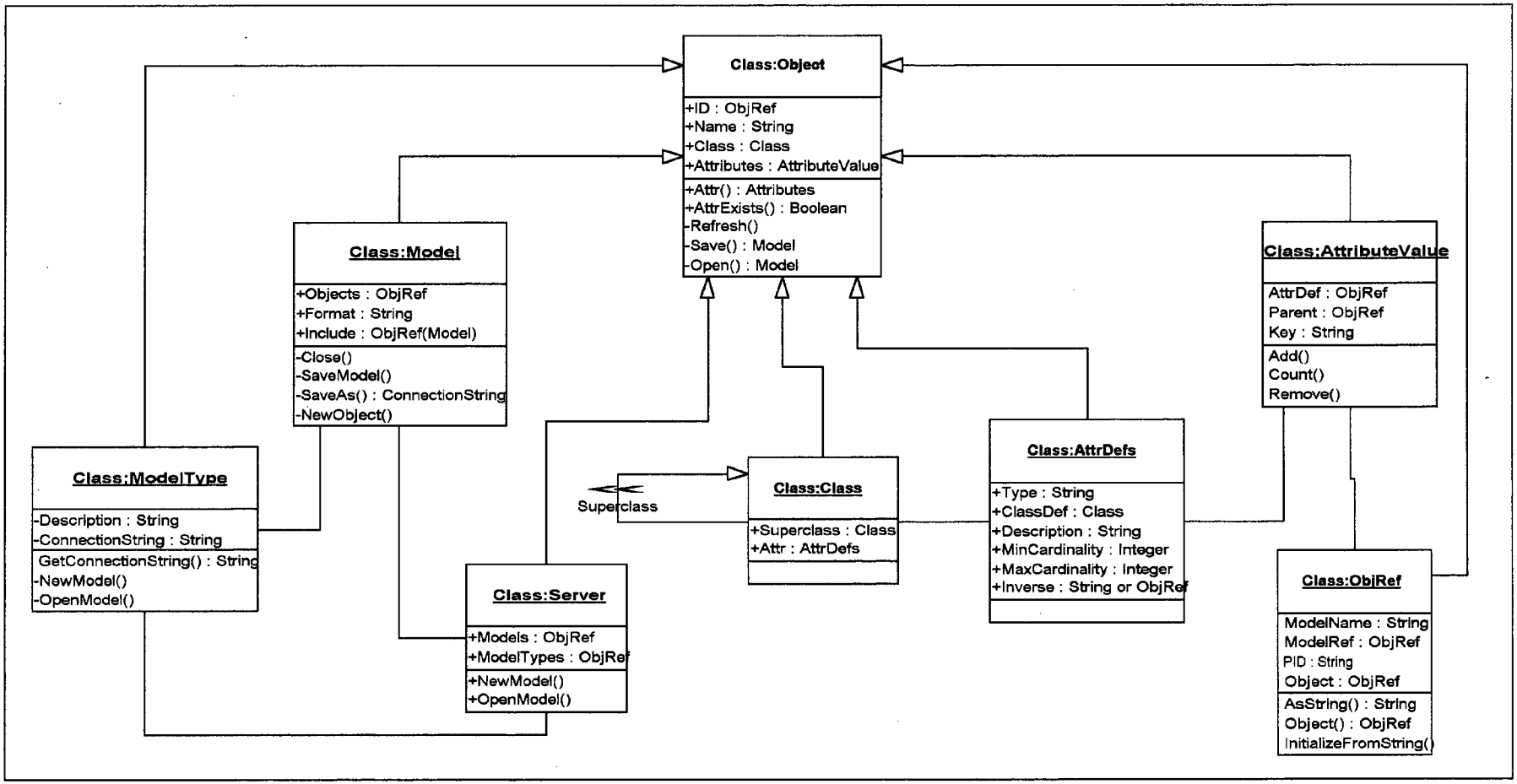

Figure 3.4 LDAI object model

An important function of this model is its ability to evolve its schema. The main reason for this is that the LDAI is being developed based on standard project data models concurrent with the ongoing development of the standard data models themselves, so the applications' underlying data structures are frequently changing and this ability allows the LDAI to keep pace with the changes. The LDAI object model implements schematic evolution by early binding only the smallest possible metamodel structure. All other metamodel constructs are built up dynamically as necessary at runtime, and changes can be made to the schema even after it has instance data associated with it. Although the LDAI will be able to maintain data consistency for most types of simple changes to existing schemas, it leaves the ultimate responsibility for maintaining data integrity during schematic evolution to the application developer and users.

It is important to note that LDAI is not intended to provide a new standard for functional interfaces between data and applications, nor is it a replacement for the SDAI. The LDAI is a development interface rather than a production interface. It is advanced as a simple and flexible tool to facilitate the quick development of prototype systems. In a full implementation of the TIP, the LDAI would be replaced with an SDAI or other data access interface.

3.1.2 Microsoft Repository

Microsoft Repository is a software component that adds an object-oriented data management layer on top of relational databases. The Microsoft Repository tool has arisen from the software development industry where it has been used for storing both software components and information about the components (metadata) in collaborative design systems. Microsoft describes its Repository as an object-oriented metadata management facility used for storing and sharing information about software systems and how they interrelate. In this role, it provides a common format for storing information about databases, reusable software components, and

assorted data warehousing tools, enabling disparate tools to share metadata and work together. (Microsoft 1999f) However, the Repository is well suited to any application area where complex object data must be stored along with object management features such as dynamic schema definition, object versioning, etc.

Repository's engine is a type-driven interpreter, that is, a user specifies an information model that consists of a set of type definitions and the repository engine automatically structures the database as required to store instances of that information model. An information model is a description of the types of data objects that are supported by an application (or a suite of applications), and the relationships that exist between those data types. Microsoft Repository stores information models as Repository type libraries (Microsoft 1999f). Models can be installed at any time and used immediately thereafter, so an application can install a model whenever needed. After an information model is installed, the repository offers operations for creating objects that are instances of the model's types, and for storing and retrieving these objects' properties and relationships to and from the repository database. (Microsoft 1999c)

The Repository engine interprets the type definitions contained in information models. Type definitions are nothing more than ordinary repository objects that have certain type-specific properties and relationships (Bernstein et. al 1999). Microsoft Repository supports the following kinds of type definitions (taken from Microsoft 1998):

1) *Class* – The class is the template from which an object instance is created. The class specifies which interfaces the object implements.
2) *Interface* – A defined set of properties, methods, and collections that form a logical grouping of behaviours and data. Classes implement interfaces and an interface may be implemented by many different classes.

3) *Property* – A scalar attribute that is defined as a member of an interface. A property has an assigned data type; for example, string, or 32 bit integer. A property is a part of the definition of an interface. A property value is an instance of a property.
4) *Relationship* – A logical connection between one object, the origin object, and a second object, the destination object. It defines which collections (on which interfaces) are connected by instances of the relationship class.
5) *Collection* – A set of repository relationships of the same relationship type that are all connected to a common source object.
6) *Method* – An invocable function that is a part of the definition of an interface.

Figure 3.5 is a contrived information model that illustrates the structure of information models in Repository. It is a simple model that supports only two classes: *Project* and *Activity*. The Project class contains two interfaces: *IProject* and *IProjectActivity*. Interface IProject describes the project, specifically the location and owner of the project, while interface IProjectActivity describes project activities that can be used by the project. Projects and activities are related to one another through the relationship *Has* which is accessible via the *ProjectActivities* collection on IProject and the *Projects* collection on IProjectActivity. In other words, a project *has* a collection of project activities and a project activity *has* a collection of projects. (The collection type definition object is used to define the cardinality of relationships so that a project may have multiple activities whereas an activity may have only one project.)

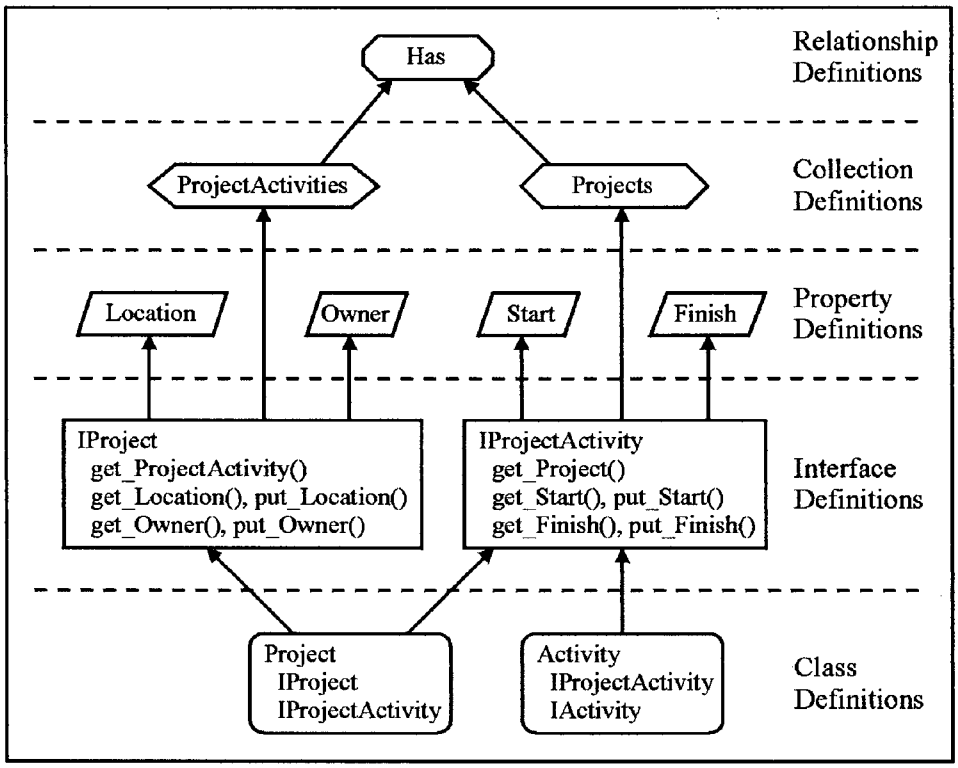

Figure 3.5 A Repository information model (after Bernstein et. al. 1999)

Note that properties and relationships specific to activities are captured by *IActivity* (not shown). The activity start and finish information are able to be exposed by the IProjectActivity interface because it contains a reference to IActivity through the Activity class.

All type definitions in an information model are instances of the classes found in the Repository type model shown in Figure 3.6. The Repository type model classes are described as instances of themselves. That is, each of the type definition objects in Figure 3.5 is an instance of the type definition classes described in Figure 3.6. For example, in Figure 3.5, there are two relationships: *Projects has ProjectActivities* and *ProjectActivities has Project*. There is an instance of the RelationshipDef class for both of these relationships. The interfaces *IProject* and *IProjectActivity* are instances of the InterfaceDef class, and so on. In this sense, the repository is self-describing. This characteristic is useful for model-driven tools, such as generic browsers (of which the TIP

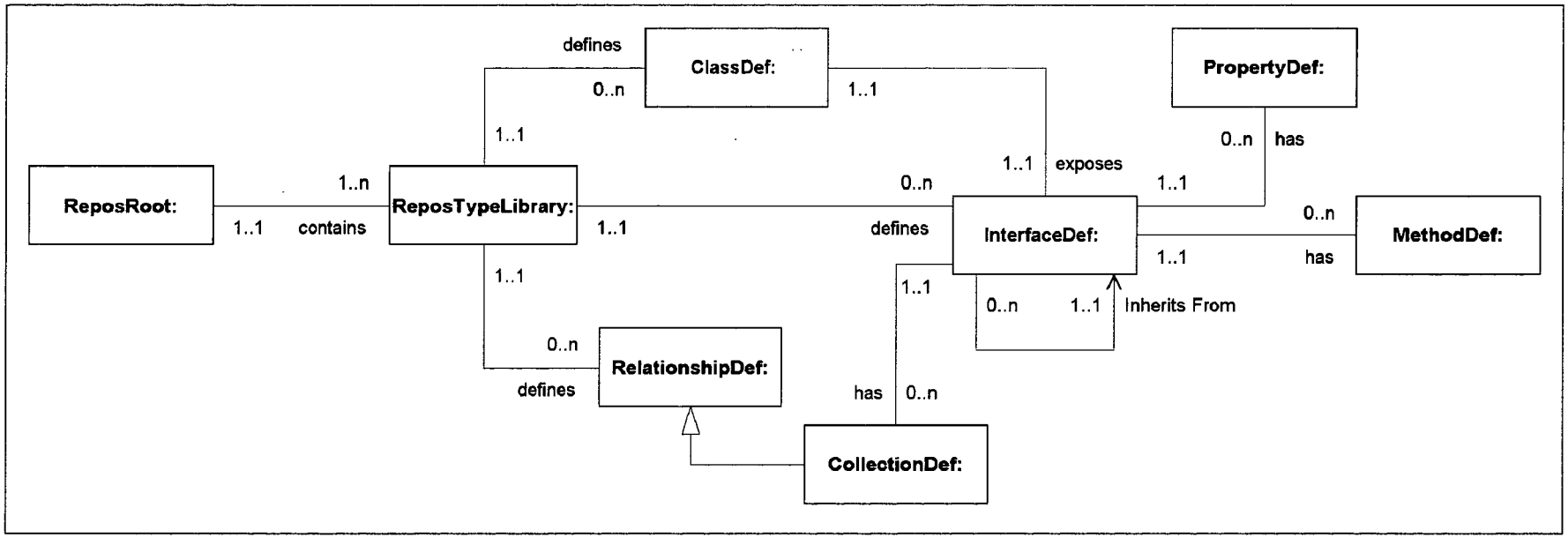

Figure 3.6 Repository Type Model

data browser is an example),that need to discover the information model at run-time and that should be able to view the repository's type model in the same way as models customised for applications. It also positions the repository to apply its own features to its own type definitions. For example, the Repository type model classes in Figure 3.6 are extensible by customers. Another example is that, when the repository engine supports versioning of type definitions, the operations to manipulate those versions will be identical to operations for manipulating versions of other kinds of objects.

Since repository instances are self-descriptive, the data they represent must also be self-descriptive. As a result, a tool is therefore able to query the contents of an information model and may use the results to adapt its processing. The interfaces of the repository completely encapsulate the stored information separating the data from the repository object. (This is illustrated in Figure 3.7. MyProject is an instance of the Project class in Figure 3.5). This makes it possible to evolve data in response to information model changes, since tools depend only on the information model, not the stored representation of data. (Microsoft 1999d)

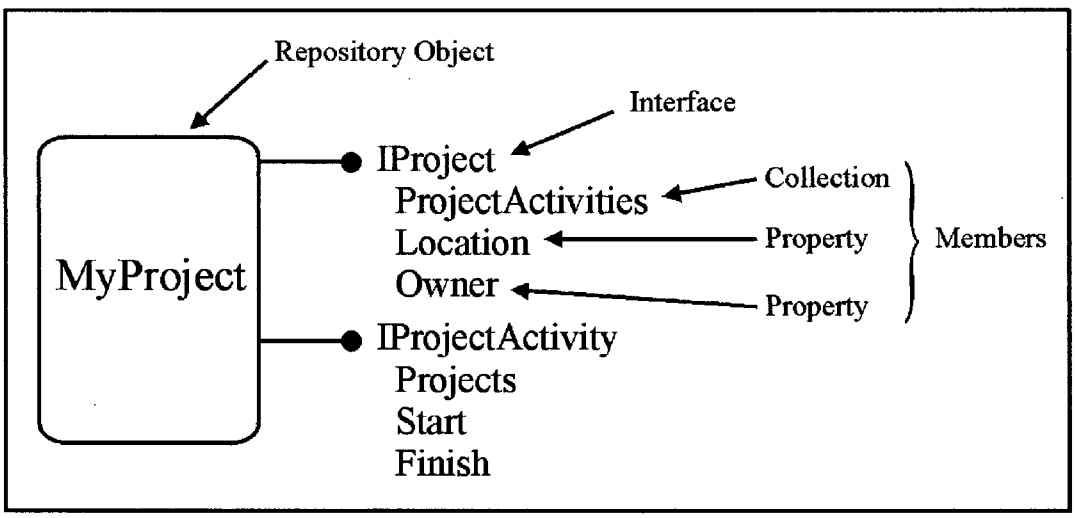

Figure 3.7 Sample Repository Object (after Bernstein et. al. 1999)

3.2 TIP Data Browser

The TIP data browser is a general-purpose user interface to the project information repository and is an integral component of TIP. Implemented in Visual Basic, this data browser can serve as a separate tool or application, similar to that of the Windows Explorer tool in Microsoft Windows. Its purpose is to allow users to enter, view, and manipulate project data. Figure 3.8 shows a typical screen of the TIP data browser. The left-hand window is the object tree. It displays all the objects that are stored in the repository in an object hierarchy. The right-hand window is the information window. When an object in the object tree is clicked, the information for that object is displayed in the information window. In this case, the information window is displaying the contents of object Activity DEF in the information window on the right hand side of the browser. The information window can display three types of data constructs: *CollectionDef*, *Property*, and *PropertyDef* objects.

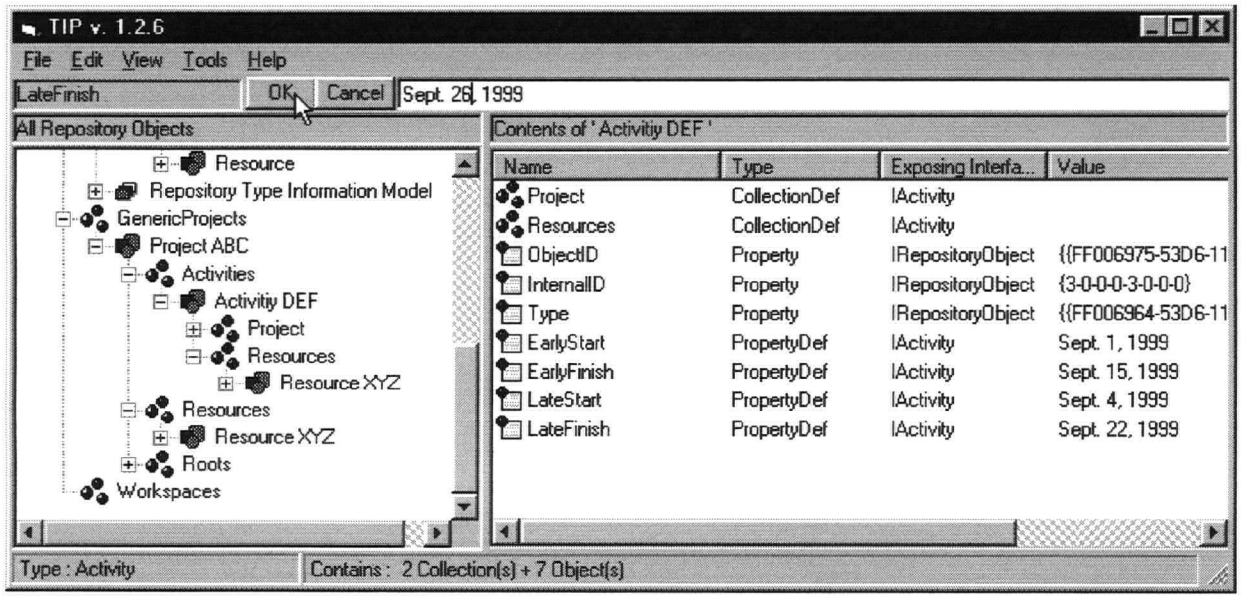

Figure 3.8 TIP data browser screen

CollectionDef's are the collections of other objects to which Activity DEF is related. Properties are system-assigned variables that uniquely identify Activity DEF. PropertyDef's are user-assigned variables that describe the state of Activity DEF. The object in question is an activity,

so the state of that activity is being described by scheduling information (early start, late start, etc.). However, any type of information can be persisted as a PropertyDef.

The TIP data browser has ability to view all available project information from various views and levels of abstraction. Project data is structured hierarchically with a detailed view displaying composite information. Views are an important concept in repository technology and are highly relevant to the subject of integration architecture. A 'view' is data presented in a way suitable for a specific task or user. In general, views are derived from subsets of data held in the repository. Base data is manipulated in some fashion to create a view, typically to create a higher level abstraction, a common requirement for construction systems. (O'Brien 1996) The most important feature of the browser is its ability to view and modify the models and schemas underlying data stored in the project repository without destroying the information in the repository. Clearly, this adds schema evolution capabilities to the TIP. Schema evolution is the ability to change a model over time. Rezgui, et. al. (1998) establish schema evolution as an important feature of an integrated data environment because building lifespans can extend over several decades or more. This allows the underlying conceptual model used by the project actors to be altered and to evolve over time, without affecting the overall consistency of the project information base. Finally, the browser can serve as a standalone application or it can be incorporated into other, more specialised applications giving users the ability to access all available project data at any given time to manipulate the data or underlying schemas as necessary.

3.2.1 Using the TIP Data Browser

The process of using the TIP data browser may be divided into three types of tasks:

1) defining an information model in Repository;
2) populating the information model; and,
3) browsing and using the data in the information model.

The following is a brief scenario presenting screen captures of how the TIP data browser is used to accomplish the first two tasks. The third type of task, browsing and using the information in the data model, is illustrated through a usage scenario in Chapter 5.1 and, therefore, will not be discussed here.

3.2.1.1 Defining an Information Model

The first step in defining an information model is to create a type information model (TIM) in the repository. The TIM articulates and defines the types of data to be stored in the repository. In this case, a model called the 'Generic Project Model' has been created. (See Figure 3.9) Once a TIM has been created, it can be populated with classes. Figure 3.9 shows three classes that have been added to the model: Project, Activity, and Resource (in mid-process of being added). Since the repository exposes a class' properties and methods through interfaces, interface objects are automatically generated for each class that is added. The default notation for an interface is the name of its class with an 'I' prefix.

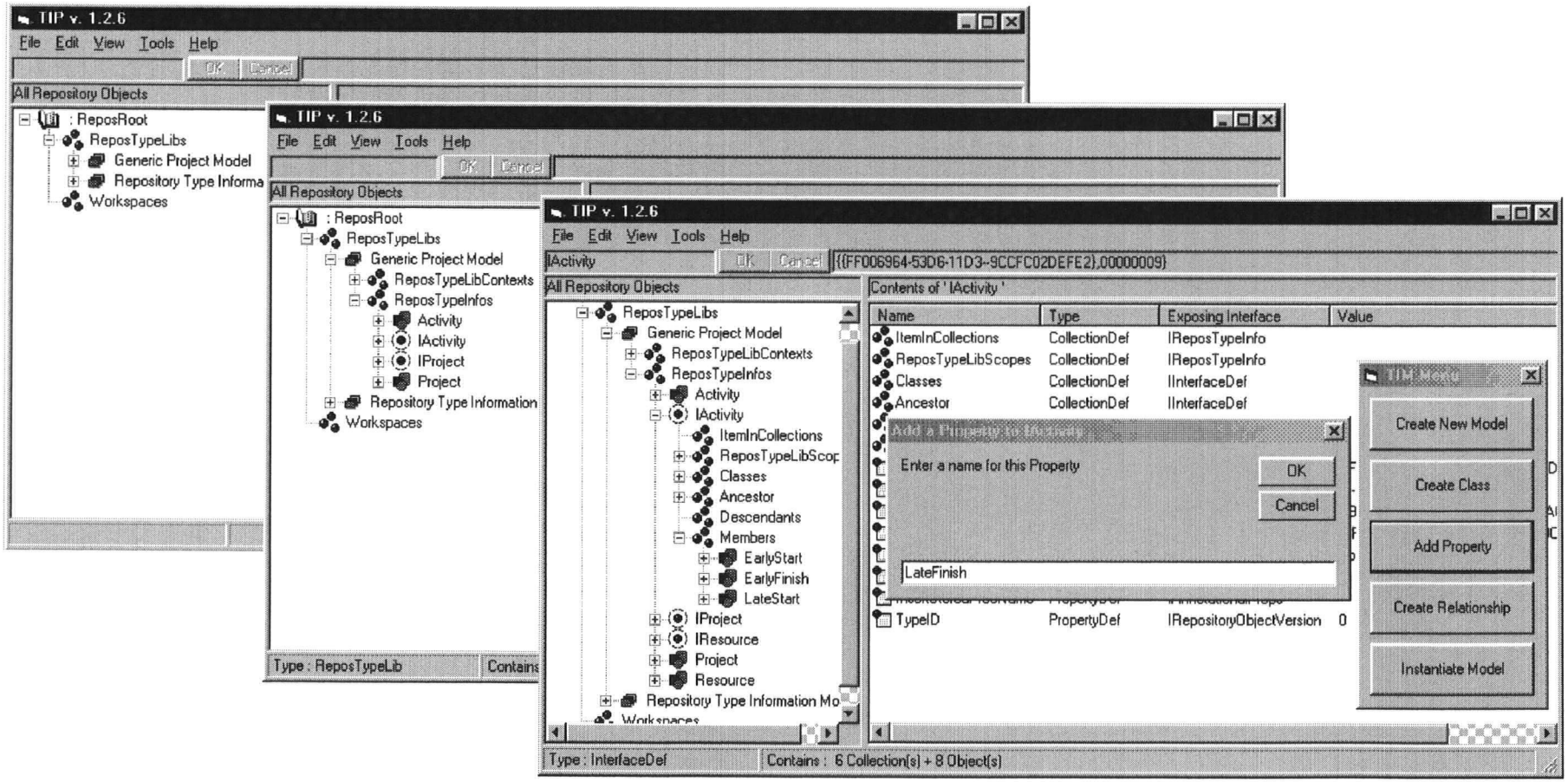

Figure 3.9 Building an information model in TIP

Once classes and interfaces have been added, the user can define the properties and methods for a call on the interface. The user may specify whether the property is of type string, integer, date, currency, binary, or time.

New models, classes, properties, and methods are all added in the same manner – by pushing the appropriate button on the TIM Menu (at the far right of Figure 3.9) and then adding the name of the model, class, or property being added. Models are defined on the Repository Root object, classes are added to the Repository Type Library and properties are defined on the interface for a given class.

Once the constituent parts of a model have been added to the repository, it is necessary to build relationships between objects to indicate associations. Since it is interfaces that expose relationships, relationships are defined on the interfaces for the associated classes. Figure 3.10 shows two existing relationships in the repository: one between Activity and Resources (called *IActivity_Resources_IResource* – in other words, activities use resources) and one between Project and Activity (*IProject_Activities_IActivity* – projects have activities). A third relationship is being defined: 'project has resources.' This is done by selecting the *Project* and *Resource* interfaces from a child window and then specifying the name of the relationship (as well as the name of its inverse relationship). In Repository, all relationship names must be unique. The notation appearing in Figure 3.10 (and used above) to specify relationship names is an arbitrary notation developed by the author to satisfy this condition.

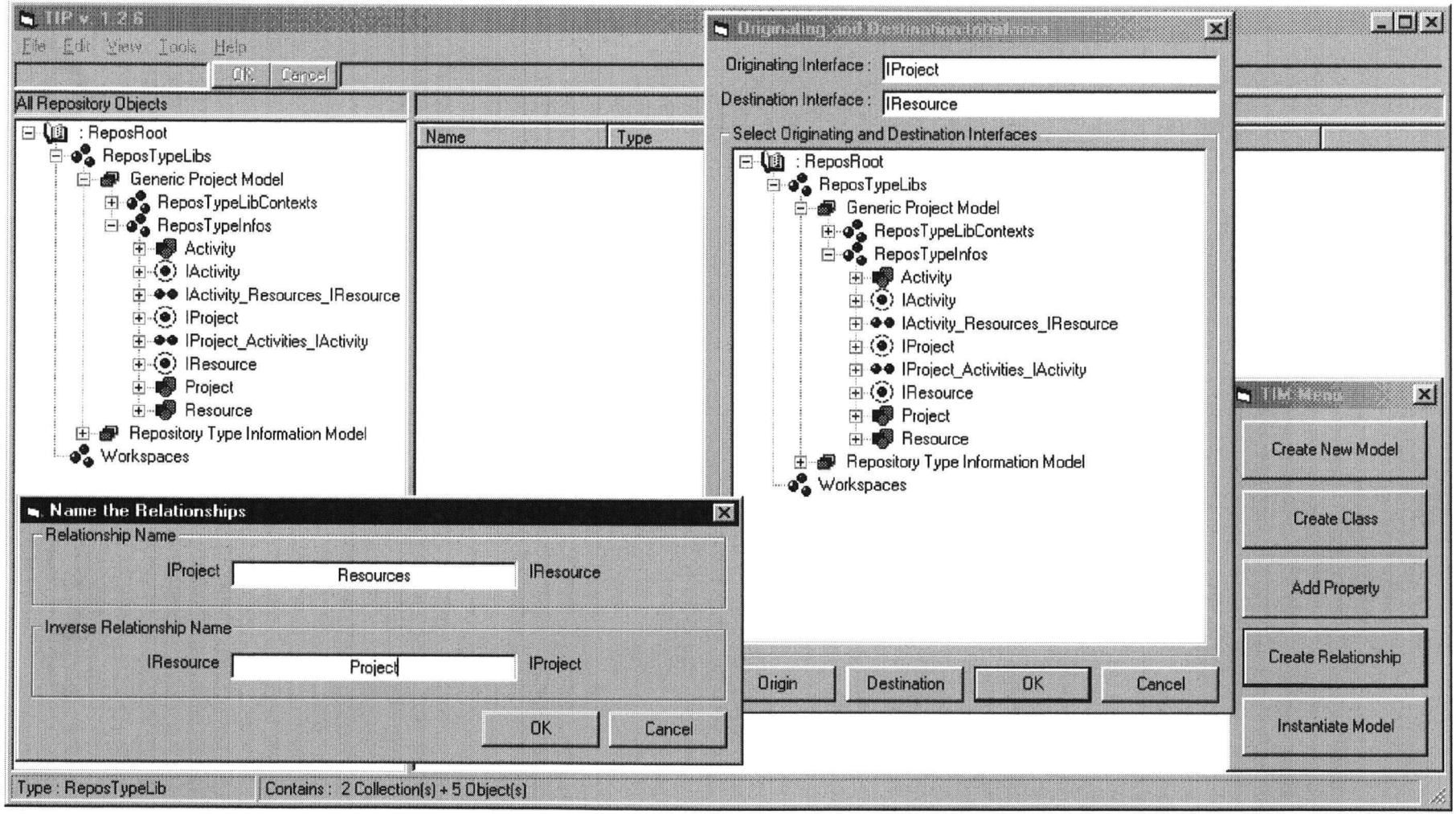

Figure 3.10 Building relationships in an information model

3.2.1.2 Populating a Data Model

Populating a Repository model may be accomplished in two ways: applications may populate models directly or the TIP data browser may be used to populate the model. Populating a model using the TIP data browser is a process of adding new objects to the model by right clicking on the object tree where the new object is to be inserted. The properties and methods of newly created objects are empty and must also be populated with data by the user. Figure 3.11 shows the instantiated model populated with some data. On the left of the figure, a user is adding a new activity object to the 'Activities' collection of the 'Project ABC' object. (Note that new objects may only be inserted into collections.) Since the relationships between objects were defined in the TIM, the model is aware of its own object hierarchy (i.e. projects have work tasks and resources, work tasks use resources). Therefore, when a user right clicks a collection to add a child object, only the appropriate child objects can be added to the object hierarchy. An information model need only be defined once and then only update as needed

In addition to adding new objects, the TIP data browser also supports functions: delete objects, rename objects, copy objects, paste relationships between objects, and edit the property values of objects. The right hand screen in Figure 3.11 shows the 'LateFinish' property of object Activity DEF being changed from Sept. 22, 1999 to Sept. 26, 1999.

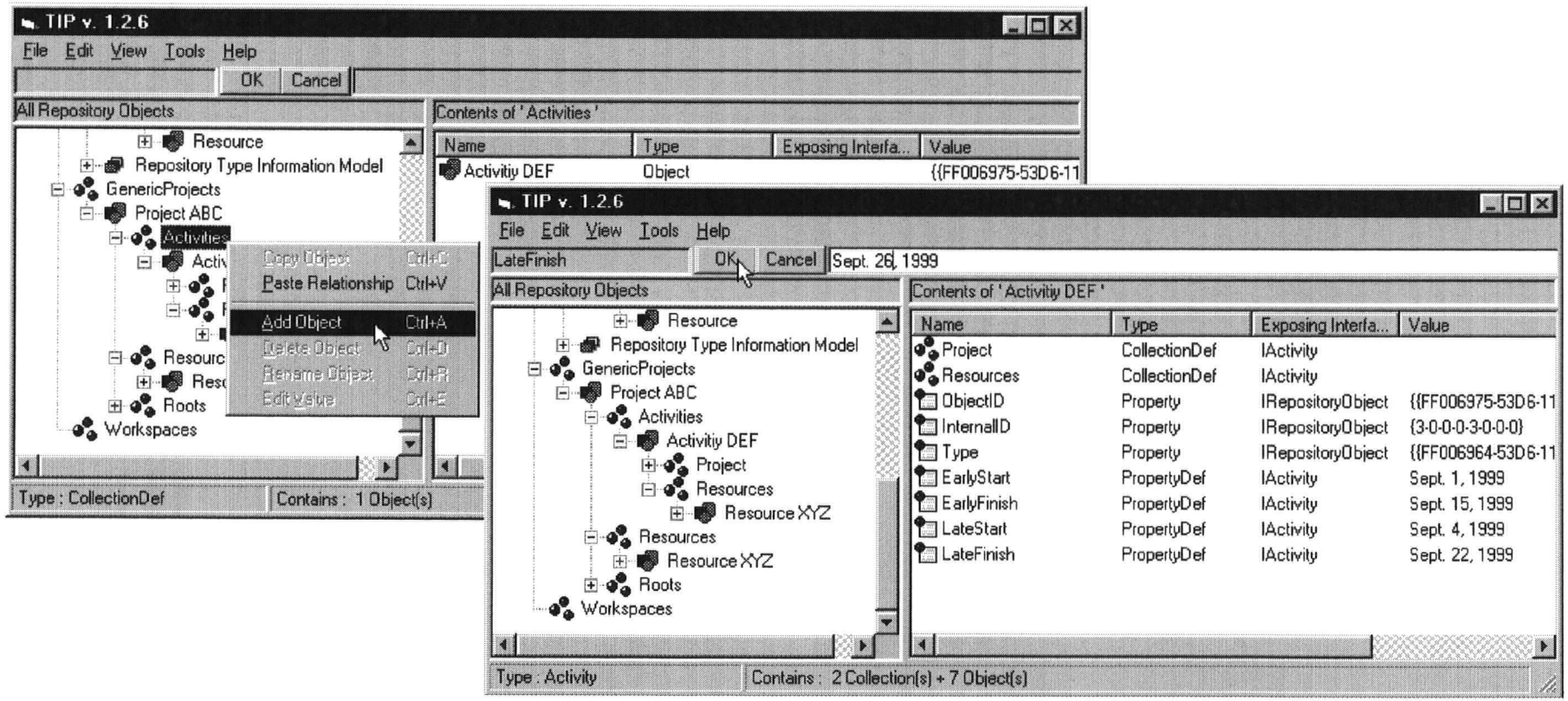

Figure 3.11 An instantiated information model

3.2.2 Caveats on the TIP Data Browser

Before continuing, some important caveats need to be mentioned about the TIP data browser. First, it is envisioned that the tasks of defining and populating data models would typically be done by system developers (such as the Construction Management Group at the University of British Columbia). While end users (i.e. project managers) are free to examine and edit the models in Repository should the need arise, it is more likely that they will only use the browser to view and update project information.

Second, the operations involved in populating Repository models using the TIP data browser are repetitious and time consuming (particularly if the models being installed are large). More typically, applications would populate the models directly through their interfaces with the Repository. This saves time and effort as it can automate the Repository operations.

Third, the TIP data browser was intended to operate on the project repository either directly or through the LDAI layer. For the sake of this project, however, the browser was implemented without incorporating any of the LDAI components into its architecture. As such, it is only capable of operating on repositories directly. Adding the LDAI components to the TIP data browser would allow it to operate on remote project repositories, or on data sets stored in other formats, as well.

Fourth, it is important to note that, although the TIP data browser interface appears almost identical to the data browser interface that Microsoft provides with Microsoft Repository, the programmatic nature of the two data browsers is substantially different. The only feature common to the two browsers is the ability to view the contents of a repository. The data and schematic editing and manipulation functions illustrated in the preceding pages are unique to the

TIP data browser and are not possible with Microsoft's browser. An example of some typical code that was used to program in the repository environment is presented in Appendix A.

4. SCSapp

SCSapp is a sample Short Cycling Scheduling APPlication. Its primary purpose is to demonstrate how an integrated data environment functions inside the TIP architecture, how data is managed, and how such a system could be integrated into the construction industry's existing information technology infrastructure. The sub-objective of SCSapp is to assist construction managers in creating lookahead schedules and to assist decision-making about work-in-place activities and materials delivery. The author has adopted the Lean Construction Institute's methodology of lookahead scheduling as proposed in Ballard (1997).

SCSapp is coded in Visual Basic and can be plugged into the TIP architecture to commit and retrieve project information to and from the project repository. The rational for developing a scheduling application as opposed to another project management tool is due to the fact that scheduling draws upon a broad range of project information (i.e., product, process, and resource information). For example, activities, sequencing, cost, the assignments of resources to processes, etc. As a result, it is expected that such a scheduling application would provide a good test of emerging IAI models of construction process-related information (although this is not the primary purpose of this thesis).

4.1 Implementing SCSapp

Figure 4.1 illustrates the architecture of SCSapp in the TIP context. SCSapp has two backends (i.e. persistence managers) implemented as part of its architecture: a repository backend and a Microsoft Project backend. The system is free to retrieve or contribute data to either backend. Project information is retrieved by the Data Access Engines via the LDAI

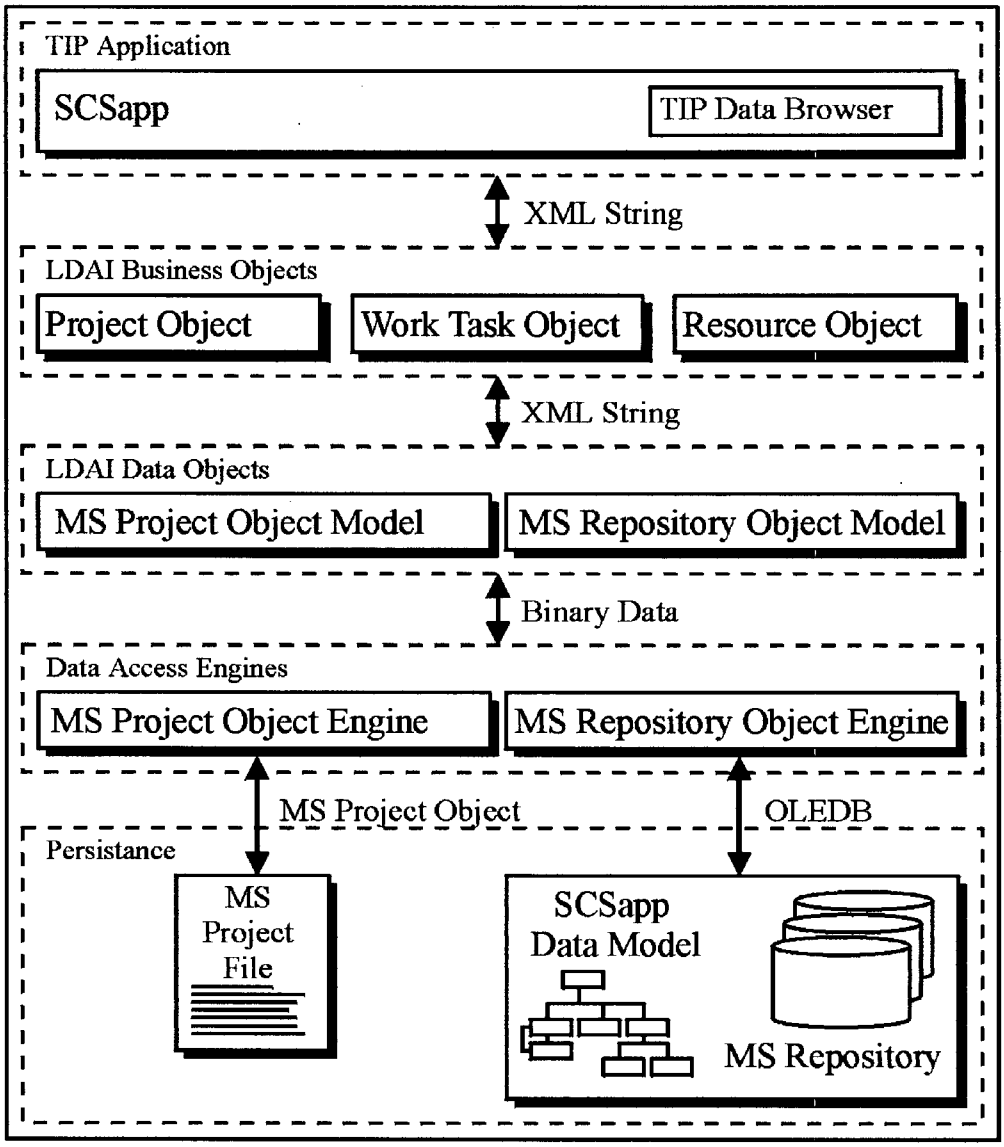

Figure 4.1 SCSapp in the TIP context

Data Objects. The Data Access Engines are the proprietary engines used by MS Project and Microsoft Repository to expose data to outside access. Generally, any attempts to retrieve and store information from either of these persistence managers must understand and use these engines. The LDAI Data Objects, MS Project Object Model and MS Repository Object Model, wrap their respective Data Access Engines to provide Project and Repository information access to the rest of the TIP (wrapping the engines makes them transparent to the rest of the system). The author implemented the MS Project and MS Repository object models as LDAI Data Objects to allow the Data Objects to communicate with the Data Engines. Once the project

information is read into the system, it is converted to an XML string and communicated to the LDAI Business Objects where the data is reconstituted into project objects, work task objects, and resource objects. Once the data has been reconstituted into objects, it is again converted to an XML string and communicated to the application level where it is interpreted. At this point, SCSapp may perform scheduling operations on the data.

The project, work task, and resource objects encapsulated in Figure 4.1 by the LDAI Business Objects are highlighted in Figure 4.2 in UML notation. These are the core objects in the SCSapp information model. They are a simplification of the IAI IFC Release 1.5.1 Object Model (see IAI 1998 for more information). Any information imported from MS Project into SCSapp is first converted to this object model and SCSapp information stored in the repository is structured according to this schema.

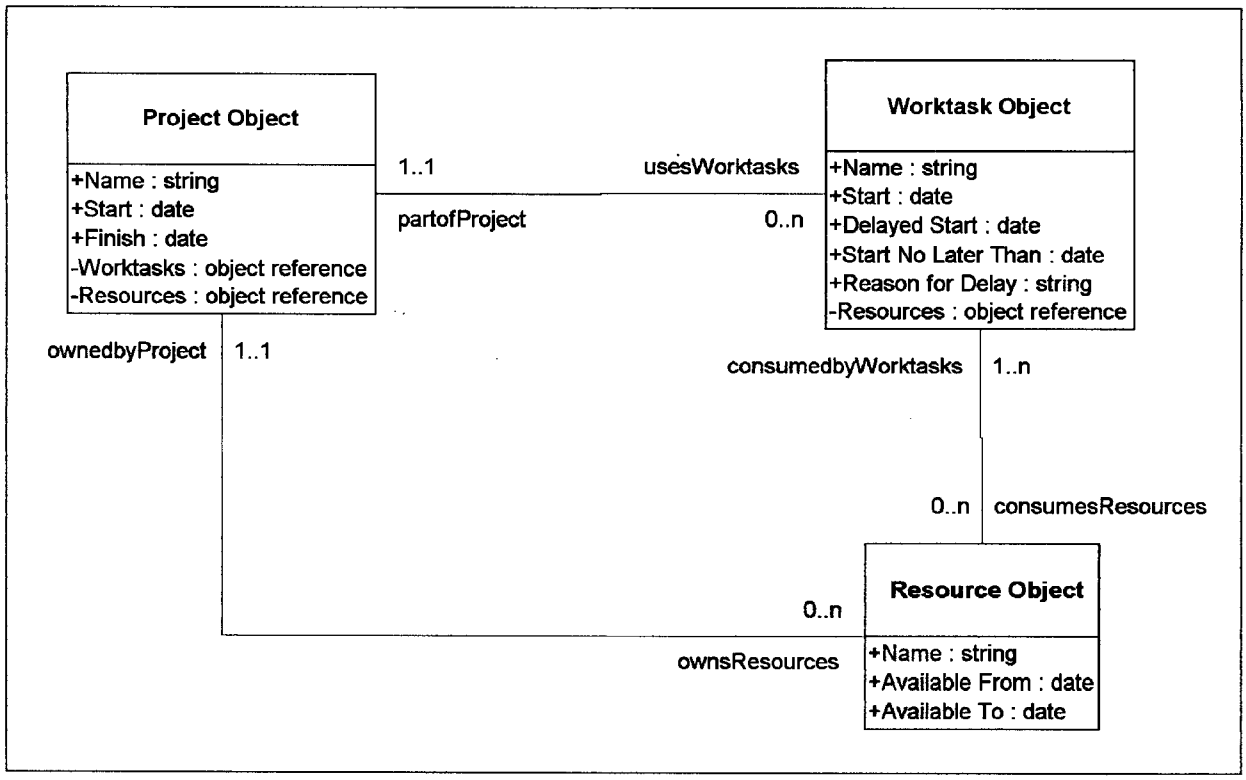

Figure 4.2 SCSapp object model

The SCSapp information model is quite simple as it represents only the most generic information necessary for a short cycle scheduling application. *Project* may contain many *work tasks* and many *resources*. *Work tasks* may use many *resources* and *resources* may be used by more than one *work task*. The project object maintains a reference to all the work task objects and all the resource objects in the schedule. Any given work task object maintains a reference to only those resource objects that it requires to complete its activity.

4.2 How SCSapp Works

4.2.1 Scheduling Methodology in SCSapp

Since SCSapp draws upon the scheduling methodology of the Lean Construction Institute (Lean) for its own scheduling methodology, a brief introduction to the Lean methodology will be presented. This explanation of the Lean methods draws from Ballard (1997) as its primary source unless otherwise referenced.

Most construction projects issue a master schedule at or near the beginning of the construction phase, extending from the beginning to the end of the project. Generally, the master schedule is used as the basis for developing a lookahead schedule. Lookahead schedules are a more detailed plan that bridges the gap from the overall project schedule to the organisation of the tasks performed at the crew level (Hinze 1998). These schedules are often called "lookahead" or "short cycle" schedules because they look ahead several weeks into the future of the project. Lookahead schedules are commonly used to focus management attention on what is supposed to happen at some time in the future, and to encourage actions in the present that cause the desired future. However, lookahead planning may also be used to:

1) shape work flow in the best achievable sequence and rate for achieving project objectives that are within the power of the organisation at each point in time;

2) match labour and related resources to work flow;

3) produce and maintain a backlog of assignments for each frontline supervisor and crew, screened for design, materials, and completion of prerequisite work at the CPM level;

4) group together work that is highly interdependent, so the work method can be planned for the whole operation; or,

5) identify operations to be planned jointly by multiple trades.

The Lean Construction Institute has been refining lookahead planning in an attempt to improve the success rate of completing tasks assigned in weekly and daily plans. Their proposed methodology is illustrated in Figure 4.3.

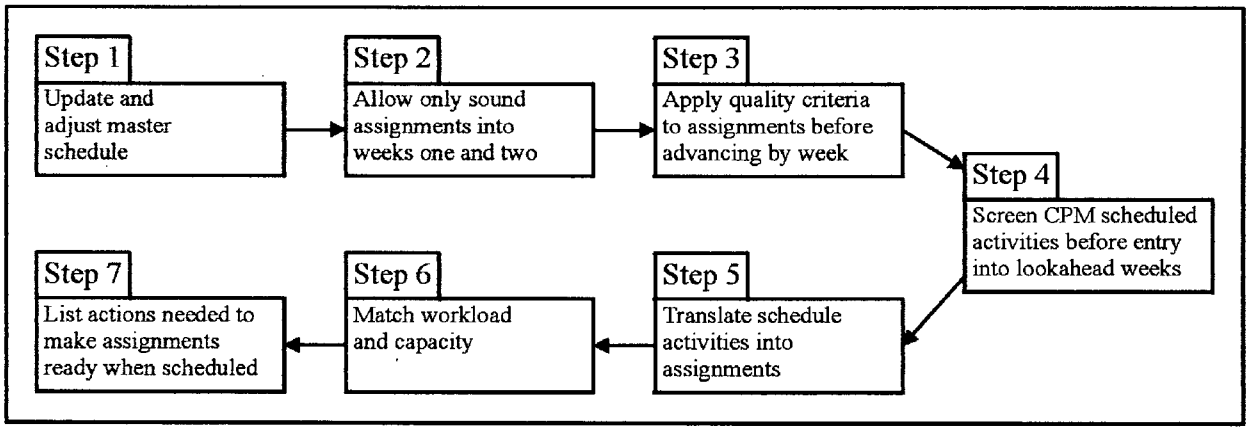

Figure 4.3 Lean Construction Institute methodology for lookahead schedules

Step 1: Enter the latest status and forecast information into the project master schedule. Adjust starts, completions, sequences, and durations accordingly.

Step2: Do not allow any assignments into week one that are not ready, except by the project management's decision. Ask the foreman if each assignment can be completed in week one, recognising that he/she may have to determine completion of prerequisite work at the item level, arrange for prework such as scaffolding, and coordinate the use of shared resources such as equipment or special tools. Allow that amount of work into week one that can be completed in the week.

Step 3: Examine the remaining weeks in the lookahead, except for the last, moving from present to future. Screen out any assignments that cannot be made ready when scheduled. Try to maintain for each crew an amount of assignments twice that which can be completed in a week.

Step 4: Identify those activities scheduled to start or complete in the lookahead week and screen out any activities that you do know can be made ready to assign when scheduled. Take into consideration the status of design, including pending charges or open issues, the availability of materials and components needed for each activity, and the probability that prerequisite work will be complete when needed.

Step 5: Translate lookahead week activities into the language of assignments, grouping highly interdependent operations that should be planned as a whole and identifying operations to be planned jointly by multiple trades.

Step 6: Calculate the earnable man-hours or otherwise quantify the labour content of the work in the lookahead week. If that amount of work falls below the amount needed to maintain schedule and if you will have the labour capacity to do that amount of work, advance work from the master schedule to the extent practical. If the resultant amount of work falls below the current workforce, reduce the workforce, or decide how to use the excess labour time. If that amount of work exceeds the current or projected workforce, decide whether or not to increase labour to accelerate progress.

Step 7: Produce a list of actions needed to make assignments ready when scheduled.

As previously stated, the purpose of SCSapp is to test and demonstrate the TIP. As such, it is necessary to make note of the fact that it is not the purpose of this research to develop SCSapp into a complete short cycle scheduling package. Therefore, SCSapp will only prototype steps three and four of the Lean Construction Institute lookahead scheduling methodology. The assumption shall be made that SCSapp only forms a small part of a larger scheduling process that

is responsible for the other steps in the Lean Construction Institute lookahead scheduling methodology.

4.2.2 Using SCSapp

SCSapp is an easy-to-use application that allows a user to look at all the work tasks that start between a set of given dates and to reschedule those tasks as necessary. Figure 4.4 is a data flow diagram detailing the overall control flow and scheduling methodology employed by SCSapp. This diagram is complemented with several screen captured images of SCSapp to help simplify the control flow and scheduling process. In the data flow diagram (Figure 4.4, item 1), the project manager (PM) first selects a project from either an MS Project source or a repository source. Opening a data source is done through an 'Open File' dialog box – an identical process to any standard Windows application (see Figure 5.3 in section 5.1 for an example). Once a project has been selected and opened, the scheduling information for that project is loaded into the system where the information is automatically reconstituted into a project object, work task objects, and resource objects (as discussed in *4.1 Implementing SCSapp*).

At this point (item 2), the PM may preconfigure the types of delays the system is to use should an activity need to be rescheduled due to a delay. Figure 4.5 illustrates this. The window on the right is the window that users see initially upon running SCSapp. From this window, users may open projects (in this case the project 'SC_Schedule.mpp' has been opened) and may configure the delays. The window on the left indicates that four delay conditions exist. The user has set the associated delay time for each condition.

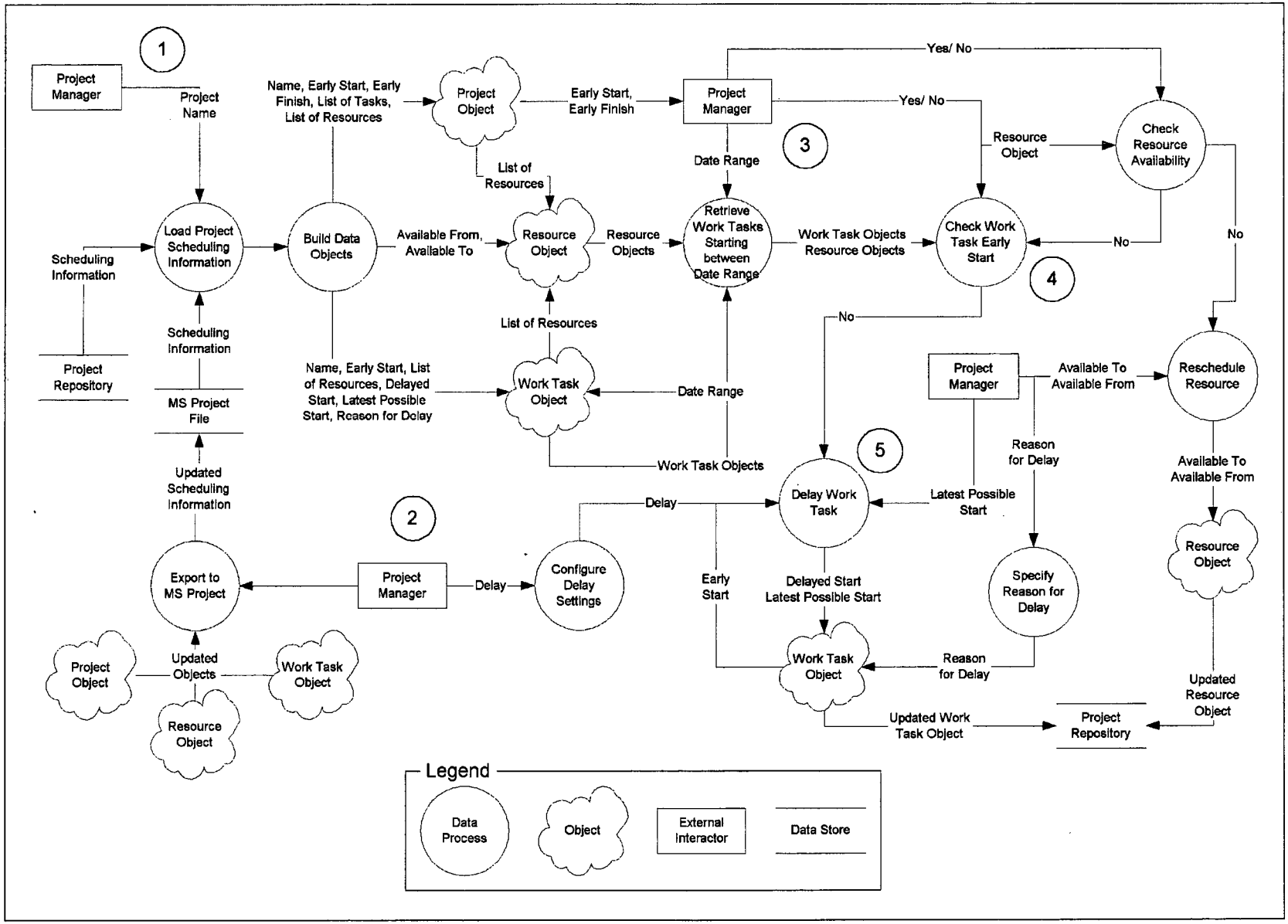

Figure 4.4 Data flow diagram of SCSapp

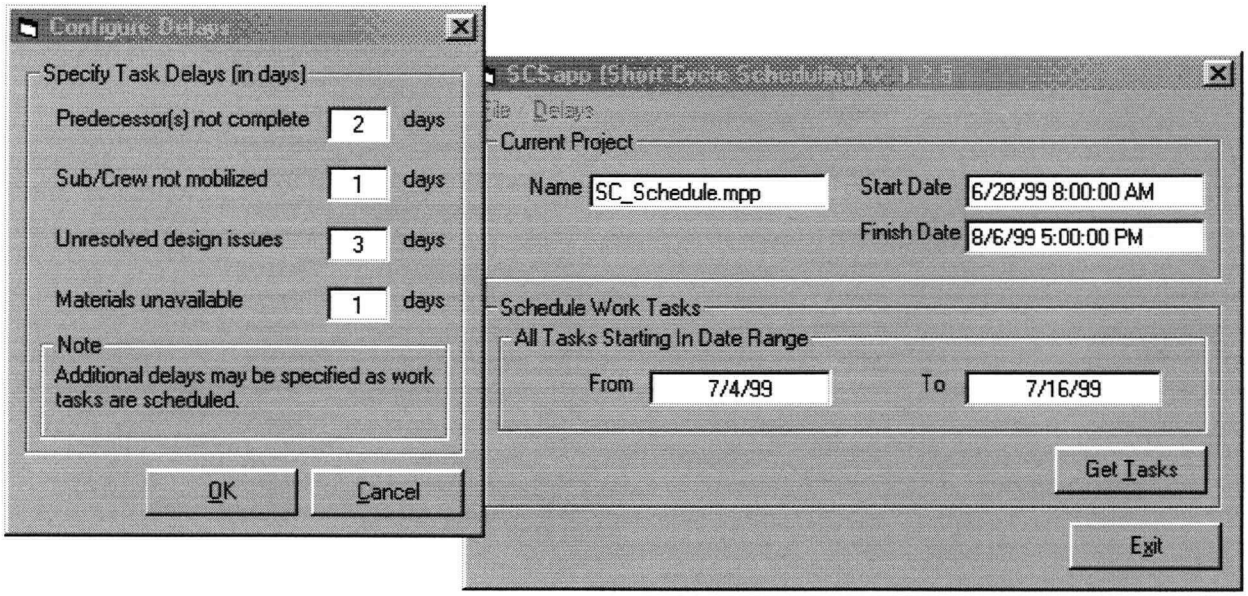

Figure 4.5 Loading a project and configuring the delays

Once the delays have been configured, the PM may retrieve all the tasks that start between two given dates (within the project's date range) (item 3). In this case, the PM is interested in the tasks that start between '7/4/99' and '7/16/99' – a two week lookahead. Retrieving tasks opens the scheduling window pictured in Figure 4.6. This corresponds to entering the data flow diagram (Figure 4.4) at 'Retrieve Work Tasks Starting between Date Range.' The scheduling window allows the user to determine if any conflicts exist in the prescheduled project information. For instance, if a resource is unavailable when a work task starts, the window raises a flag next to the offending resource in the appropriate work task. These appear as the triangular flag in Figure 4.6.

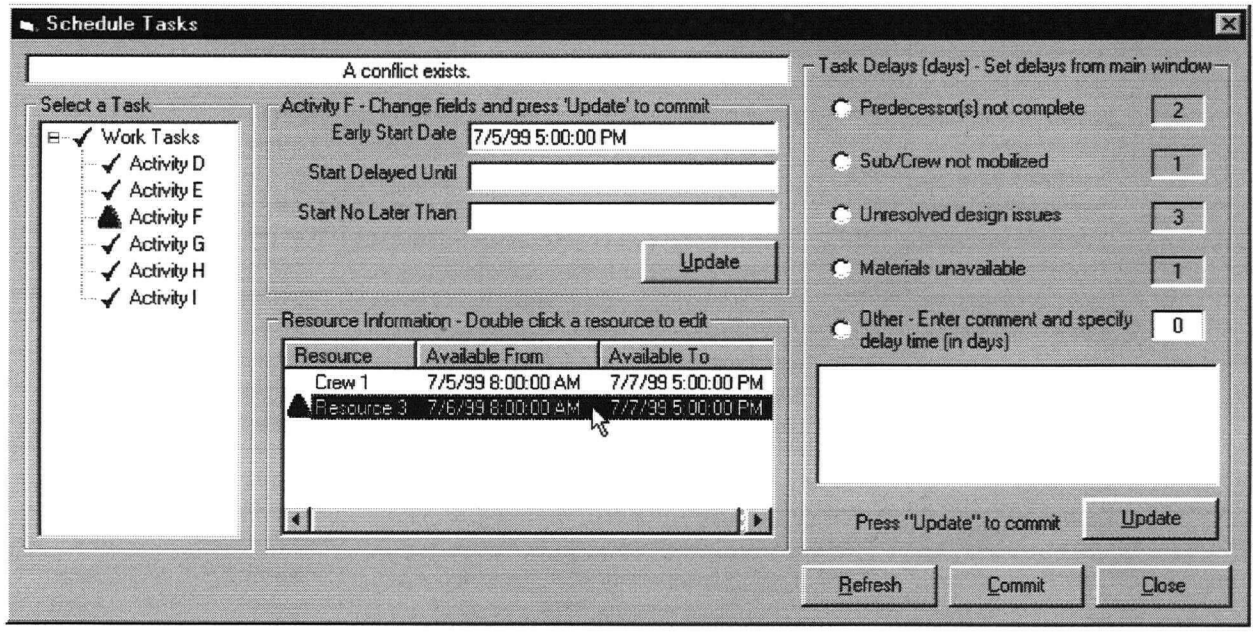

Figure 4.6 SCSapp scheduling interface

It is the responsibility of the PM to check the work tasks (item 4), entering information which is not present in the prescheduled information that may yet delay an activity. The delays that were configured in Figure 4.5 appear at the far right of the scheduling window. Should an activity be delayed, the PM can simply check the appropriate cause of delay. If the delay is not present in the check list, the manager may enter an additional delay specifying both the time the activity is to be delayed and the reason for the delay. This corresponds to the 'Delay Work Task' node in the data flow diagram. Once the delay is entered, the work task object is updated and is committed to the repository.

Delaying an activity (item 5) may necessitate the delay of the resources used by that activity. This can be done by double clicking the appropriate resource in the scheduling window and rescheduling its 'Available From' and 'Available To' dates. As shown in the data flow diagram, this updates the Resource object which is then committed to the repository. The repository is automatically updated as the PM makes changes to the scheduling information. MS Project files are only updated when the PM exports the data to MS Project.

4.2.3 Implementation Issues in SCSapp

There are a number of implementation issues in TIP not addressed in SCSapp that bear mentioning. First, SCSapp does not yet have the ability to export project information to MS Project. Since an MS Project exporter would essentially be the reverse of the MS Project importer already coded by the author, an exporter is not a necessity to prove the open, distributed, and modular nature of TIP. It would, however, serve to make SCSapp a more complete application. Should the MS Project backend be completed (i.e. through the addition of the exporter) a mechanism would be needed that ensures that all project information, regardless of its source, is synchronised with the repository. In other words, the project information in MS Project and the project information in the system repository must be equivalent in order to ensure that all users are working from the same version. SCSapp does not currently have such a mechanism implemented. As a result, it falls upon the user to be vigilant in maintaining the integrity of the data by ensuring that all applications are working off of equivalent versions. Repository has the ability to version information (and maintain copies of the old versions) but this feature has not been exploited in SCSapp.

Second, the LDAI middleware component of SCSapp did not take shape as originally intended. The SCSapp object model (see Figure 4.2) is part of the LDAI business object layer but it is a static model. It was intended to be evolution-capable but this was not possible as the LDAI is not yet fully developed and implemented.

Third, project information is not communicated between the tiers as XML strings (as shown in Figure 4.1), information is communicated as binary data. This obviously impacts upon the distributed nature of SCSapp. However, adding code to SCSapp to convert the binary data to XML strings is not considered a major barrier. It was decided to forego this while SCSapp is still

an early prototype since the object models SCSapp is using are continually changing. This would have necessitated the recoding of the XML converters (late bound objects which must contain these object models in order to function) several times during the prototyping stage. Once the object models have been finalised, then the XML converters may be easily added.

5. Discussion and Implications

5.1 Discussion

This section is designed to provide the reader with a practical illustration of how the TIP Data Browser and SCSapp would function if they were implemented and deployed in a construction environment and how they could potentially aide a project manager. A use case scenario (Figure 5.1) has been developed using the Jacobson Objectory Method. (See Fowler (1997) for more information on using the Jacobson Objectory Model.)

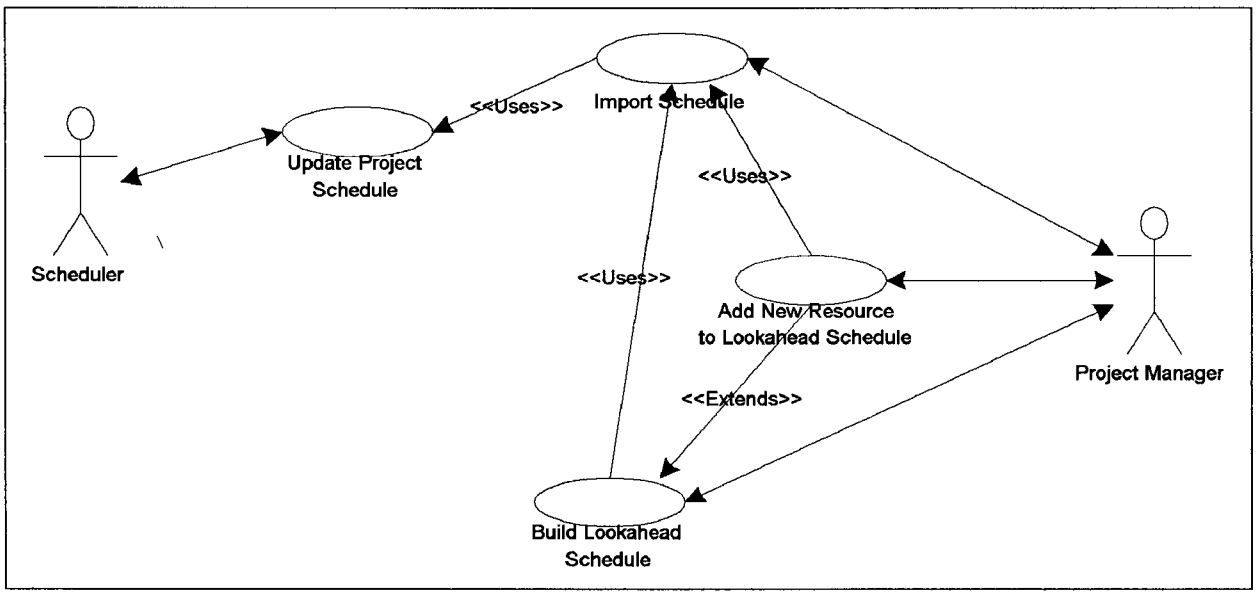

Figure 5.1 Use case scenario

The scenario is quite simple: a project manager (PM) needs to produce a lookahead schedule. The PM imports the schedule (from MS Project) and builds the lookahead schedule (using SCSapp). However, the PM discovers that an activity will be delayed due to a resource that is unavailable. The PM decides to view all the available project data (through the TIP data browser) to determine if there is another resource that could be used as an alternative.

The construction data used in this usage scenario is a fictitious set of data that was generated in MS Project. It is a very simple schedule containing just enough activities so that float exists on

some of the activities. Figure 5.2 is a screen capture of the construction data in MS Project displayed in a Gantt chart.

An assumption is made that this construction data is part of a larger project. Therefore, the data presented in Figure 5.2 does not represent the schedule for the entire project as a whole. This schedule may represent a trade or subcontractor working on the project.

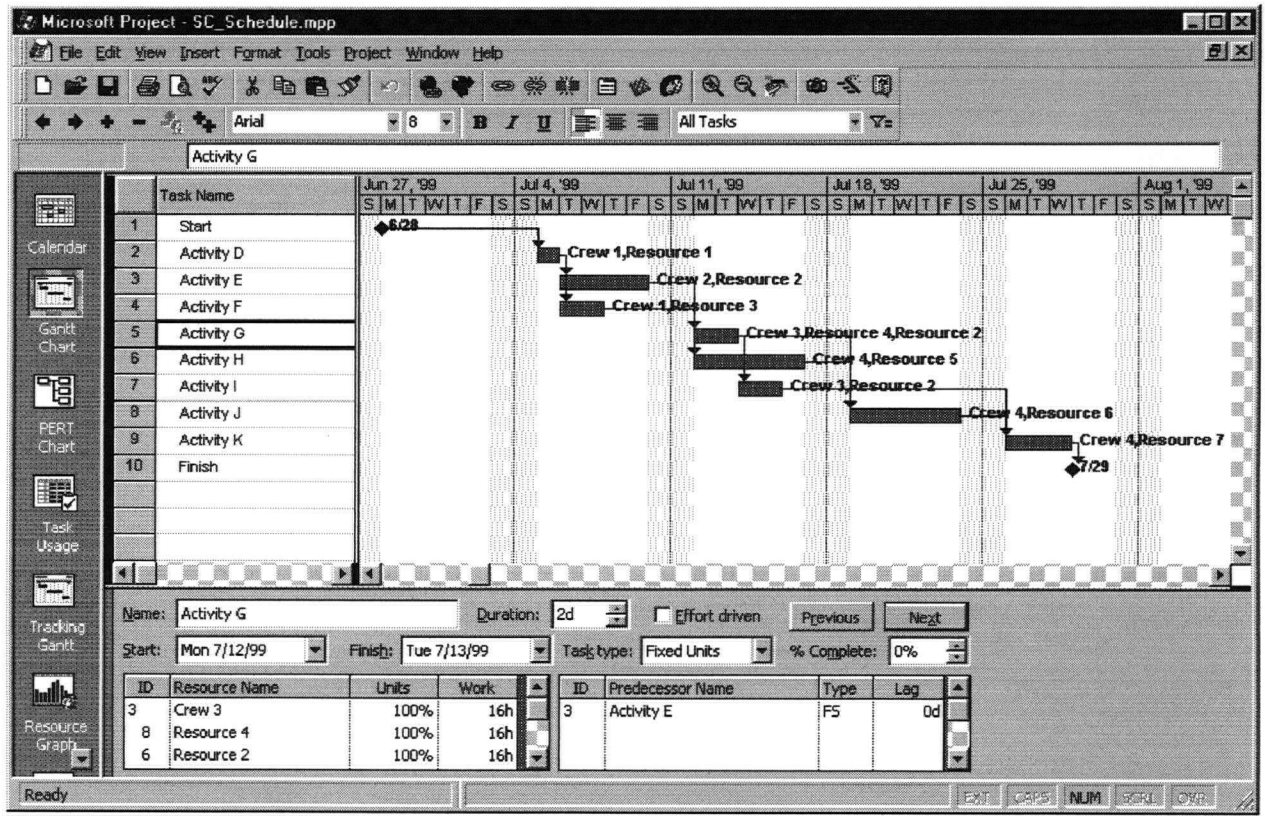

Figure 5.2 Sample construction data in MS Project

The PM begins by importing the schedule from MS Project by selecting the desired project through an 'Open File' dialog box (Figure 5.3). A more advanced version of SCSapp that supported XML strings would have the additional option of allowing the PM to open the schedule by specifying an Internet address that contained the schedule.

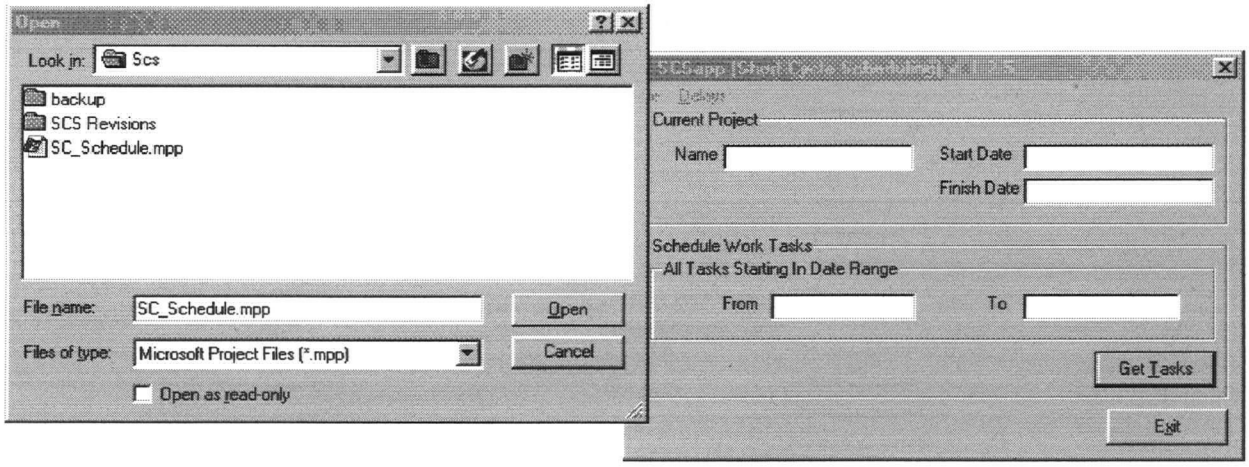

Figure 5.3 Importing a schedule

Once the project has been imported the PM may preconfigure the types of delays (and the length of those delays) the system is to use should an activity need to be rescheduled due to a delay. Figure 4.5 (in section *4.2.1 Scheduling Methodology in SCSapp*) illustrated this process so it will not be repeated here. Once the delays are preconfigured, the PM selects the weeks for which (s)he wishes to build the lookahead schedule. In this case, the PM is interested in all activities starting in the two-week period between '7/4/99' and '7/16/99' (Figure 5.4).

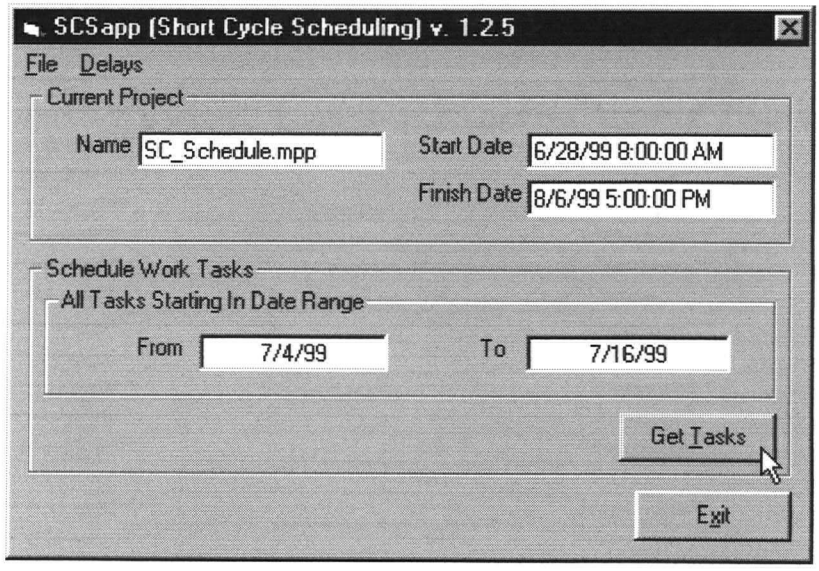

Figure 5.4 Selecting the tasks for the lookahead schedule

Once the lookahead period has been specified, the scheduling window is displayed for the PM. It consists of the list of all work tasks starting between the dates '7/4/99' and '7/16/99' (far left of

Figure 5.5). In this case, a flag tells the PM that there is a scheduling conflict in *Activity F.* Clicking on Activity F exposes its early start date as well as the resources it uses. The flag has been raised by Resource 3, which will not be available to Activity F until the day after its scheduled start.

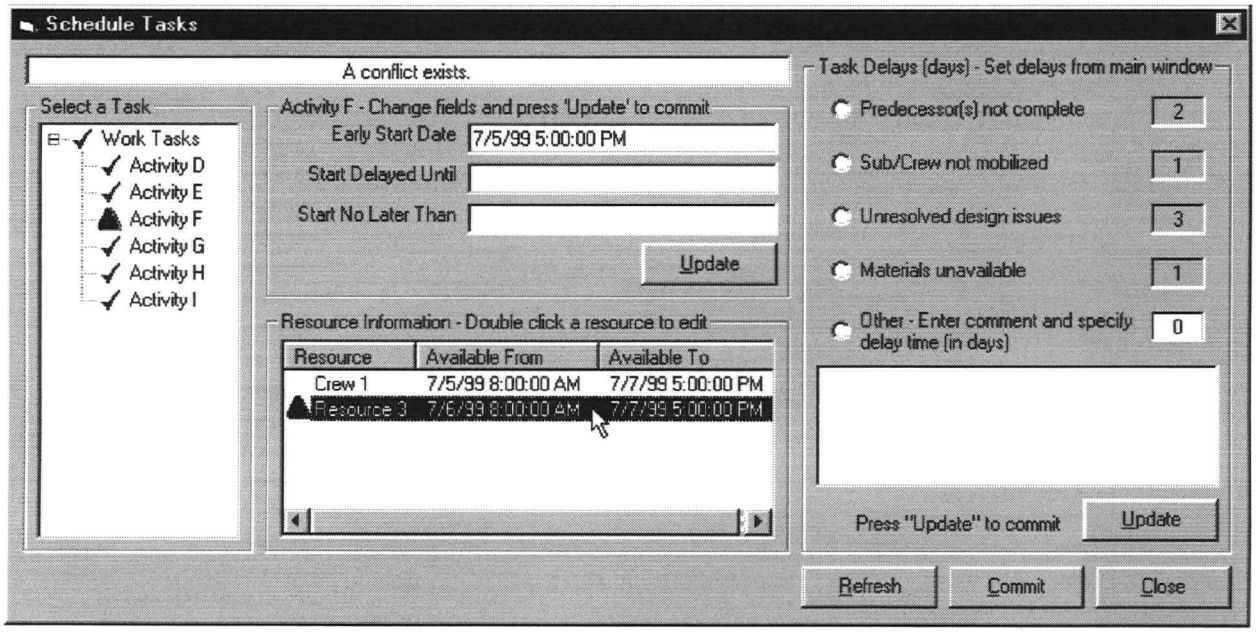

Figure 5.5 A conflict in the prescheduled information raises a flag

At this point there are two options open to the project manager. The first is to simply delay Activity F by a couple of days until the resource becomes available. The PM can easily do this by clicking on a Task Delay (far right of Figure 5.5) to delay the activity. Were the PM to do this, the outcome may look like something similar to Figure 5.6.

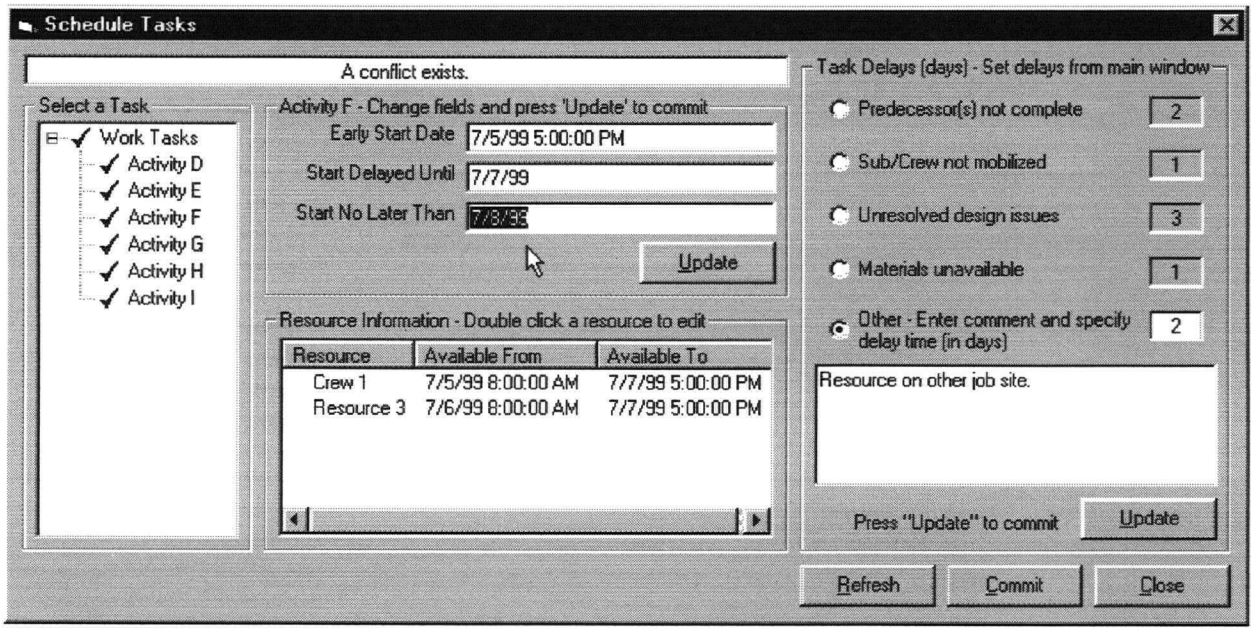

Figure 5.6 Delaying Activity F

In this instance, the PM has delayed the activity by selecting the 'Other' Task Delay and specifying that the activity is to be delayed for two days as the resource is on another job site. This automatically updates the activity start date, delaying it until '7/7/99' (upper centre of Figure 5.6). The PM has taken a further step and also specified that the activity is to start no later than '7/8/99.'

A second option is available to the PM however. Rather than delay Activity F, the PM can enter the project repository and search through the project information for other means to start the Activity F on schedule. The PM enters the project repository with the TIP data browser. Figure 5.7 shows the browser with Activity F and its associated resources (Crew 1 and Resource 3).

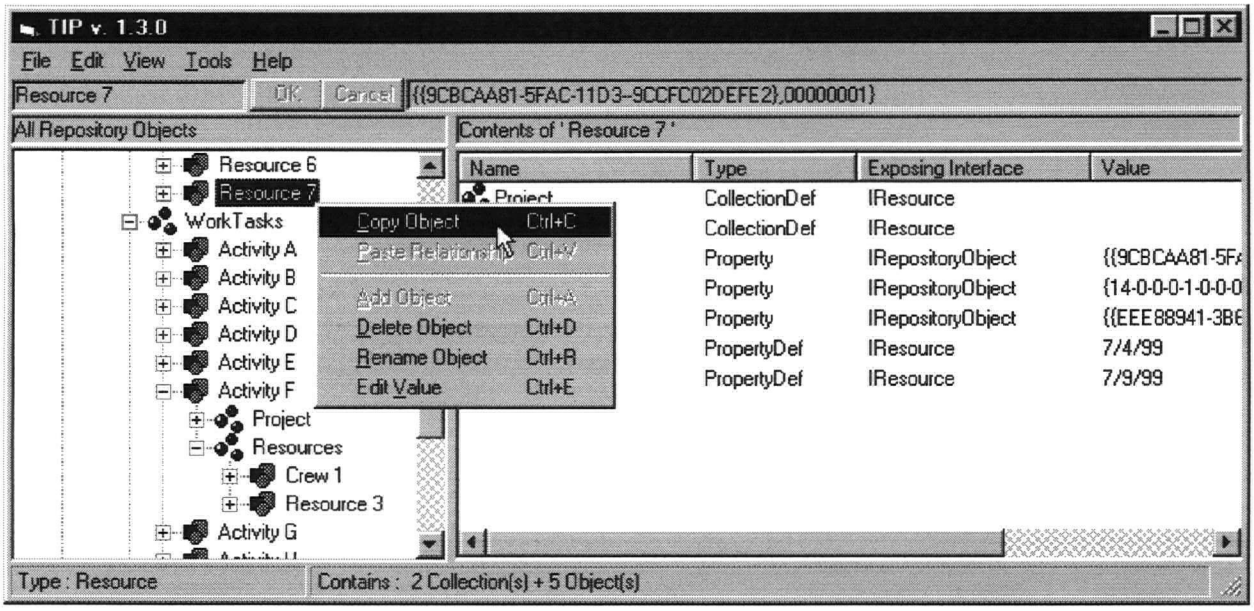

Figure 5.7 Browsing the project data

After browsing the work tasks and resources in use on the project, the PM realises that there is an additional resource, Resource 7, which is available from '7/4/99' to '7/9/99' and which is not in use. By right clicking on the resource, the PM may copy it and paste it into the 'Resources' collection of Activity F (see Figure 5.8).

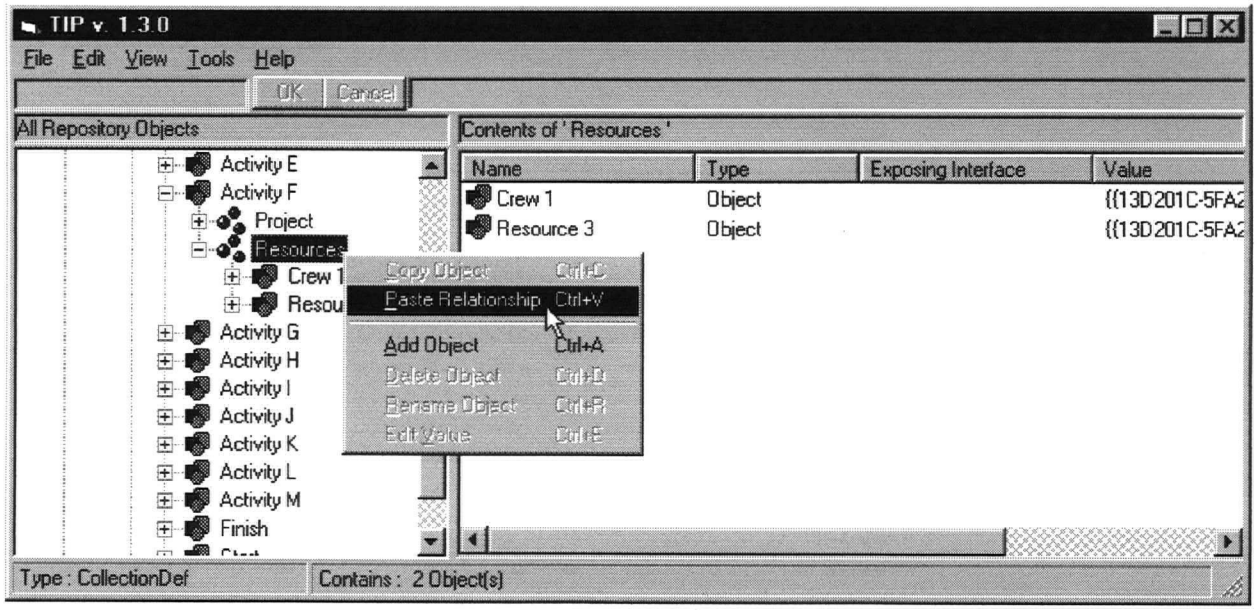

Figure 5.8 Pasting a new relationship

Recalling the SCSapp object model from Figure 4.2 in *4.1 Implementing SCSapp*, any relationship between a work task and a resource is of type 'Work Task Uses Resource' or 'Resource Is Used By Work Task' (the inverse relationship). The result is an Activity F that now has the additional resource 'Resource 7' (Figure 5.9).

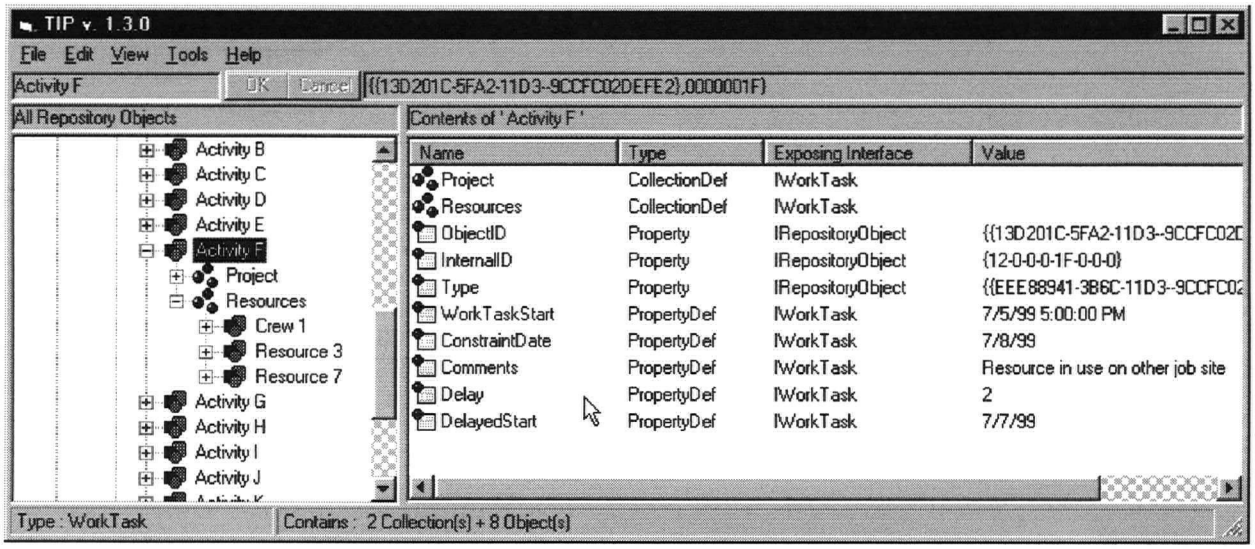

Figure 5.9 New Activity F

The PM could then edit Activity F, deleting the delayed start information from the activity. Once this is done, the activity will start on time using Resource 7 instead of Resource 3. The PM also has the choice of scheduling Resource 7 just long enough until Resource 3 becomes available, or using Resource 7 for the duration of Activity F and rescheduling Resource 3 to be used elsewhere on the project.

In addition to the review of resource availability, the PM can review each activity for a variety of other possible causes of delay such as: completion of predecessor activities, mobilisation of work forces, finalisation of all design issues, etc. Although SCSapp does not perform any specific analysis to support these reviews, it does provide a checklist of a number of them (along with an "others" option). If any of these issues arise, SCSapp can add an appropriate delay to the planned activity start time and can record the information about the delay in the project model.

5.2 Implications

This section discusses how the TIP environment has addressed the TOPS characteristics of comprehensiveness, integration, and flexibility with a window into how the TOPS model may contribute to the development and use of next generation CIC systems and environments.

5.2.1 Demonstrating the TOPS Model

Although TIP is an early prototype of elements of the TOPS approach, it nonetheless demonstrates that a semantically rich CIC environment capable of supporting project views and data versioning in a modular, distributed, open architectural framework is well within the grasp of current software technology. TIP also illustrates that any number of unique or custom interfaces can be developed for any enterprise and these interfaces can share common data with other applications and need not rely on the use of their own proprietary data formats or structures. Additionally, interfaces need not be complex, expensive, or time-consuming to develop either as this project was developed entirely in Visual Basic, a very easy to learn and accessible programming environment.

That MS Project was able to plug-in to TIP demonstrates another important potential of the TIP architecture: that integration of an enterprise's information assets may proceed with little disruption to current practices and the benefits can be immediately recognised at each step of the integration process. The only potential barrier to interoperability with existing software applications is the need to know the underlying object models that drive an application. If the object model is unknown, it will be more difficult (but not impossible) to build interfaces that allow applications to plug-in to the TIP architecture. However, given the current industry efforts to develop standard data models (take IAI for instance) many companies (AutoCAD being one)

have been willing to publish the object models underlying their systems while other companies such as Microsoft distribute the object models underlying their products as a matter of course.

Finally, the importance of supporting multiple construction views or perspectives cannot be underestimated. By supporting views, TIP provides an environment that supports the data needs of any and all project participants from a single data source. (While the views constructed in the SCSapp and TIP data browser in this thesis were simple by nature, they still point to the ability to construct more complicated views). Participants will be able to work from up-to-date versions of project information. The repository maintains previous versions of project information. Versions allow a project to track its history from conception to decommissioning. Any piece of data from any point in time may be recreated to suit the needs of the users.

5.2.2 Application of Repository Technology to CIC Systems

Repository technology was developed by the software industry to support the ongoing development and modelling of software. Although the software domain is substantially different from the construction domain, the issues being addressed by CIC are similar to the issues the software industry are trying to address through the use of repository technology, namely: consolidating and managing information assets, interoperability between software tools, reusing components, developing information models, and managing multiple versions of the same data. All these factors make repository technology an excellent candidate for persistence management in CIC environments.

Having said that, the application of repository technology in the development of CIC environments remains an innovative approach to the integration and consolidation of construction project information. A literature search revealed that most researchers are using

more traditional types of object-oriented or relational databases for persistence management (see the CIC systems cited in chapter 2.1 for examples). And, whereas many of the CIC efforts undertaken of late (such as those outlined in *2.1 Current State-of-Practice CIC Systems*) have focused primarily on achieving integration through document management or project collaboration (i.e. project communications), repository technology represents an excellent opportunity to pursue the complete integration and consolidation of project information. Additionally, this author discovered that repository technology also provides an excellent foundation for the development of CIC environments in terms of supporting many of the key technologies envisioned as necessary components of CIC systems (shared data, project views, versioning services, etc.).

As an added benefit, repository technology is being heavily invested in by software corporations (as evidenced by MS and others (see Microsoft 1999e)) as the future of software modelling and development. As a result, the construction industry may contribute the bulk of its research and development effort towards the application of repository technology in the construction environment and towards the development of standards to be used to model construction products and processes. The software industry will drive the development of repository technology.

6. Results, Recommendations & Conclusions

6.1 Results

The challenge of this research was to develop and prototype a CIC environment/architecture that would underlie and support applications developed in the TOPS context. At the beginning of this thesis, a group of objectives were set out to achieve this challenge. These objectives have been addressed in different ways:

1) *To explore the implementation issues relating to the TIP*

   At issue in the implementation of TIP was the development of an open, modular, and flexible system architecture that allows project information to be consolidated yet shared in a distributed framework. Project models needed to be extensible and the architecture had to be generic enough to allow any number of applications of different type to plug-in to the architecture. The solution developed to address these issues was the development of the multi-tiered LDAI architecture based around Microsoft Repository as the persistence manager. The LDAI provided the generic interfaces needed for applications to plug-in to the system while the development of the TIP data browser provided the extensibility functions necessary to maintain and extend the project models and information in the repository. The development of the SCSapp short cycle scheduling application demonstrated that the challenges presented by the TOPS model were not insurmountable.

2) *To describe the project environment that the TIP model will create in terms of process and data analysis and by using software diagramming techniques*

   A system design for TIP was developed as described in section 3.1. This design was tested through a usage scenario as described in section 5.1.

3) *To conceptualise and implement a seamless, integrated data architecture based on a common product data model that consolidates project information*

An integrated data architecture was constructed using Microsoft Repository. Interaction with the data environment was through the TIP data browser, which constituted the bulk of the work in developing the data architecture. It was initially envisioned that the data environment would be populated with IAI information schemas. This did not turn out to be the case since it was very time consuming to enter the IAI schemas through the TIP data browser interface. Nonetheless, the TIP prototype is capable of accepting IAI information schemas. An EXPRESS backend that could interpret the IAI schemas and build them in Repository automatically would greatly simplify this process.

4) *To demonstrate the usefulness of such a data environment using two additional implementation prototypes: the TIP data browser; and SCSapp*

The TIP data browser and SCSapp development were documented in sections 3 and 4. The applications were tested in a use case scenario in section 5.1.

The sub-objectives of the TIP data browser were addressed as follows:

1) *To use emerging repository technology to store project data*

Microsoft's repository technology was investigated and then implemented as the persistence manager for TIP. This is described in section 3.1.2.

2) *To provide experience in implementing components in repository architecture and in implementing generic user interfaces*

The author, previously inexperienced in repository environments, gained much experience and insight into the design and development in the repository environment through the programming of the TIP data browser. The author gained much experience

into multi-tier architectures and object-oriented systems through the programming of both the TIP data browser and the SCSapp.

3) *To provide a consolidated view of all project information and to build new views and abstractions of project data*

The TIP data browser allows users to navigate all project information in a repository regardless of the origin or application of that information. Sections 3.2.1 and 5.1 provide simple examples of building new abstractions and views of data. Section 3.2.1 illustrates how the TIP data browser is used to build up an information model (i.e. a view) from scratch while the usage scenario in section 5.1 shows how the consolidated view of the project assisted the project manager in rescheduling an activity.

5) *To add editing and long-term maintenance functionality to the project information base*

The TIP data browser has been implemented with a variety of functions that allow project data to be edited. These functions include: adding objects, deleting objects, renaming objects, copying objects, pasting relationships between objects, and editing the property values of objects. This is described in section 3.2.

6) *To view and edit the schemas and models underlying the project information*

The TIP data browser has the ability to restructure the metamodels underlying the data stored in the repository. Any change in the structure of the metamodels is dynamically and immediately reflected in the structure of the project information itself. This is described in section 3.2.1 and 5.1

The SCSapp also had the following sub-objectives:

1) *To provide experience into designing and implementing generic project management applications and interfaces*

Much experience in developing applications and interfaces was gained through the development of SCSapp. In particular, the author gained a tremendous appreciation for the difficulties associated with designing and implementing intuitive interfaces.

2) *To provide experience in building multi-tier system architectures*

The multi-tiered structure of SCSapp presented several conceptual difficulties for the author during the design phase. However, as development progressed, the author gained an appreciation for the need to separate systems into tiers. Separating systems into tiers allows each tier to address a different aspect of the system. Data objects handle all the data access needs. Business objects handle all the logic in the system that is common to all applications in the environment. User interfaces are responsible for the functionality specific to individual applications only. Programming the multi-tier system in SCSapp gave the author much experience into how object-oriented systems take shape and how behaviours and data is segregated.

3) *To assist project managers in developing short cycle schedules*

The SCSapp provides project managers with a short cycle scheduling tool that may assist them in developing lookahead schedules. However, since it was not the principle objective of this research to develop a complete short cycle scheduling package, the SCSapp is a rudimentary tool.

6.2 Recommendations

While the TIP prototype is largely functional (particularly the TIP data browser), it is still a proof of concept prototype of the TOPS model. If it is to be a fully compliant TOPS environment, the following improvements must be made to future versions:

1) *Greater distributed functionality*

The TIP prototype framework is currently limited to operating only locally or on local area networks. It does not yet possess the fully distributed nature (i.e. Internet or World Wide Web functionality) envisioned in TOPS. In order to accomplish this, it remains necessary to include XML components in the LDAI. Additional components to serve the system via the Internet or World Wide Web must also be added to the system. This technology is well established and well supported and stands to greatly increase the flexibility and potential of the TIP by providing communications abilities to users regardless of physical location.

2) *Dynamic LDAI layer*

The LDAI did not take shape as it was originally intended to in this thesis. The LDAI was initially expected to form a dynamic middleware layer. However, the LDAI models implemented were static. The rational behind a dynamic middleware layer was to allow it to keep pace with changes made to the schemas underlying the data layer. A static LDAI layer must be reprogrammed manually each time changes to the repository data schemas are initiated. This requires much time and effort and also requires tight, well commented code to avoid mistakes and confusion. Given that the TIP is envisioned to be populated with IAI schemas, and the fact that these schemas are still being developed, it would save much effort if the equivalent of the TIP data browser were developed for the LDAI layer to maintain and manage its components.

3) *Data versioning*

The TIP failed to exploit some of the capability of Microsoft Repository. Among the functionality remaining unexploited is the ability to maintain multiple successive versions of any piece of data stored in the repository. Given the number of revisions construction documents may go through on any given project, this function becomes a necessity. Not

only will it ensure that users are working from the same generation information, but it will also allow the history of a project to be reconstructed and its evolution documented.

4) *Repository technology must be pursued*

Finally, repository technology must be pursued as a potentially viable option for the persistence management functionality of CIC systems. Given the overlap in the objectives between the software industry in developing repository technology and the construction industry's needs in a persistence manager, repository technology may provide the foundation for the potential to push CIC environments into the next generation of development.

5) *Further validation of TOPS must be pursued*

The TOPS models of integration architecture proved to be effective guidelines for the development of a prototype integrated architecture. However, it is important to note that TOPS represents only a single approach to integrated systems and that TIP makes up only a small portion of the TOPS model. As such, more research is needed to prove the validity of the overall TOPS approach.

6.3 Conclusions

The purpose of this thesis was to investigate and explore how the architecture of CIC systems should evolve in the future to add project management functionality to the construction industry. The main goal of this research was to implement and test the viability of a prototype CIC architecture based on the Total Project Systems models of integration architecture developed by Froese et. al. (1997). The approach was to develop a TOPS Implementation Prototype. This prototype was a multi-tiered architecture with Microsoft Repository at the core providing the persistence management functions. It was successfully implemented and a generic application (a short cycle scheduling application) was constructed to test the potential of the prototype. The

system performed well given its prototype nature. And, while the TOPS Implementation Prototype does not necessarily prove that the overall TOPS model is the best approach to CIC systems, the author's experimentation nonetheless concludes that the TOPS model of integrated architectures is an effective construct to use as the basis for the development of future integrated architectures.

References

Ballard, G. (1997) Lookahead Planning: The Missing Link in Production Control. Presented to the 5th Annual Conference of the International Group for Lean Construction, Griffith University, Gold Coast, Australia, July, 1997.

Bernstein, P., Bergstraesser, T., Carlson, J., Pal, S., Sanders, P., and Shutt, D. (1999). Microsoft Repository Version 2 and the Open Information Model. [online] http://msdn.microsoft.com/repository/technical/whitepapers.asp, [accessed 13 August 1999]

ISO (1995) Industrial Automation Systems and Integration - Product Data Representation and Exchange Part 22: Standard Data Access Interface. ISO 10303-22.

Hinze, J. (1998) Construction Planning and Scheduling. Published by Prentice-Hall Inc., Toronto, Canada, pp. 270 - 282.

Fowler, M. (1997) UML Distilled: Applying the Standard Object Modelling Language. Published by Addison-Wesley Longman Inc., Don Mills, Canada, pp. 43 - 52.

Jägbeck, A. (1998) IT Support for Construction Planning. Ph.D. Thesis, Royal Institute of Technology, Stockholm, Sweden.

Fischer, M. and Froese, T. (1996) Examples and Characteristics of Shared Project Models. ASCE Journal of Computing in Civil Engineering, Special section on Data, Product, and Process Modelling, Vol. 10, No. 3, pp. 174 - 182.

Fischer, M. and Kunz, J. (1995) The Circle: Architecture for Integrating Software. ASCE Journal of Computing in Civil Engineering, Vol. 9, No. 2, pp. 122 - 133.

Froese, T. (1999) Integrated, Model-Based Project Management Systems Implementation Workshop Presentation. Presented to the Construction Management Group at the University of British Columbia, Vancouver, Canada, June, 1999.

Froese, T., Rankin, J., and Yu, K. (1997) Project Management Application Models and Computer-Assisted Construction Planning in Total Project Systems. The International Journal of Construction Information Technology, Vol. 5, No. 1, pp. 39 - 62.

IAI (1998) IFC Object Model Industry Foundation Classes - Release 1.5.1 Model Reference Documentation. International Alliance for Interoperability [online] http://iaiweb.lbl.gov [accessed 30 August 1999]

International Systems Group (1997) Middleware – The Essential Component for Enterprise Client/Server Applications. Middleware White Paper [online] http://www.isg-inc.com [accessed December 1998]

Mace, S., Flohr, U., Dobson, R., and Graham, T. (1998) Weaving a Better Web. Byte Magazine, March 1998, pp. 58 - 68.

Microsoft (1998) Microsoft Repository SDK 2.1. [online] http://www.microsoft.com/repository [accessed 15 September 1999]

Microsoft (1999a) Integrated Metadata Management. [online] http://msdn.microsoft.com/repository/whatis/metadata.asp, [accessed 16 August 1999]

Microsoft (1999b) Microsoft's Goal in Developing a Repository. [online] http://msdn.microsoft.com/repository/whatis/goals.asp, [accessed 16 August 1999]

Microsoft (1999c) Microsoft Repository Engine. [online] http://msdn.microsoft.com/repository/whatis/engine.asp, [accessed 16 August 1999]

Microsoft (1999d) Model Exchange via Repository Engine. [online] http://msdn.microsoft.com/repository/scenarios/modelex.asp, [accessed 16 August 1999]

Microsoft (1999e) History of Microsoft Repository. [online] http://msdn.microsoft.com/repository/whatis/history.asp, [accessed 03 September 1999]

Microsoft (1999f) General Q & A. [online] http://msdn.microsoft.com/repository/prodinfo/faq.asp, [accessed 10 June 1999]

deMonsabert, S. and Lemmer, H. (1997) Consolidated object Repository for Civil Engineering (CORCE). ASCE Journal of Computing in Civil Engineering, Vol. 11, No. 1, pp. 70 - 73.

O'Brien, M.J. (1996) A Strategy for Achieving Data Integration in Construction. The International Journal of Construction Information Technology, Vol. 4, No. 1, page 21 - 34.

Rankin, J., Froese, T., and Waugh, L. (1999) Exploring the Application of Case-Based Reasoning to Computer-Assisted Construction Planning. Proceedings of the Durability of Building Materials and Components 8: Service Life and Asset Management, Vol. 4 Information Technology in Construction: CIB W78 Workshop, pp. 2526 - 2536.

Rankin, J., Froese, T., and Waugh, L. (1998) The Functionality of Computer-Assisted Construction Planning. Proceedings of the 1998 Conference of the Canadian Society for Civil Engineers, Halifax, NS, June 10-13, 1998. Vol. 1, pp.119 - 128.

Rankin, J., Froese, T., and Waugh, L. (1997) Computer Assisted Construction Planning (CACP) in the Context of Total Project Systems (TOPS). Proceedings of the 1997 Conference of the Canadian Society for Civil Engineers, Sherbrooke, Quebec, May 27-30, 1997. Vol. 2, pp. 2-41 to 2-50.

Rao, G., Grobler, F., Ganeshan, R. (1997) Interconnected Component Applications for AEC Software Development. ASCE Journal of Computing in Civil Engineering, Vol. 11, No. 3, pp. 154 - 164.

Rezgui, Y., Cooper, G., and Brandon, P. (1998) Information Management in a Collaborative Multiactor Environment: The COMMIT Approach. ASCE Journal of Computing in Civil Engineering, Vol. 12, No. 3, pp. 136 - 144.

Rezgui, Y., Cooper, G., and Vakola, M. (1999) An Innovative Infrastructure for Inter-Working Between Dissimilar EDM Solutions. Proceedings of the Durability of Building Materials and Components 8: Service Life and Asset Management, Vol. 4 Information Technology in Construction: CIB W78 Workshop, pp. 2537 - 2546.

Russell, A. and Froese, T. (1997) Challenges and a vision for computer-integrated management systems for medium-sized contractors. Canadian Journal of Civil Engineering, Vol. 24, No. 2, pp.180 - 190.

Sandakly, F., Kloosterman, S., Ferreira, P., and Poyet, P. (1998) PerDiS: Persistent Distributed Store for Virtual Enterprise Concurrent Engineering. Proceedings of the CIB Working Commission W78, pp. 457- 468.

Stumpf, A., Ganeshan, R., Chin, S., and Liu, L. (1996) Object- Oriented Model for Integrating Construction Product and Process Information. ASCE Journal of Computing in Civil Engineering, Vol. 10, No. 3, pp. 204 - 213.

Teicholz, P. and Fischer, M. (1994) Strategy for Computer Integrated Construction Technology. ASCE Journal of Construction Engineering and Management, Vol. 120, No. 1, pp. 117 - 131.

Underwood, J., Alshawi, M., Auuad, G., Child, T., and Faraj, I. (1999) The Dynamic Development of Design Element Specifications via a Product Supplier Database Web-site. Proceedings of the Durability of Building Materials and Components 8: Service Life and Asset Management, Vol. 4 Information Technology in Construction: CIB W78 Workshop, pp. 2629 - 2639.

Yu, K., Froese, T., and Vinet, B. (1997) Facilities Management Core Models. Proceedings of the 1997 Conference of the Canadian Society for Civil Engineers, Sherbrooke, Quebec, May 27-30, 1997. Vol. 2, pp. 2-195 to 2-204.

Appendix A

This is a piece of sample code taken from the SCSapp application. It is presented to give the reader an understanding of how to program (late bound) in repository. Please note that this code was selected for its simplicity and accessibility to the layperson. This code was selected because it is not as complex as the code that was used to program the TIP data browser (which was programmed early bound). However, it may help the reader to review the Microsoft Repository Software Development Kit (Microsoft 1998) documentation to grasp a better understanding of the concepts involved in Repository programming. The code examples are transcribed in Courier font and are indented. Explanations of the code are transcribed in New Times Roman font and precede the code they explain.

This is a subroutine that takes Project, Resource, and Work Task Objects (in the form of a collection) from SCSapp and persists them in Repository. It requires that the collections of Project, Resources, and Work Tasks be passed to the subroutine to work.

```
Private Sub PopulateRep(ProjCol As Collection, ResCols As Collection,
WTCols As Collection)
```

In order to write information into Repository, interactions with the Repository must be bracketed within the scope of a transaction. (Microsoft 1998) The following statement opens a transaction.

```
Repos.Transaction.Begin   ' Begin a Repository transaction.
```

The necessary variables are dimensioned. Since we are creating repository objects to represent the objects in SCSapp, a repository object variable must be dimensioned (in this case, the 'Project' repository object) to be placed into the repository. All repository objects in repository must have unique Object ID's, including the Project object being created. The Object ID for the Project object will be stored in the variant 'OBJID_Project.'

```
Dim ProjectName As String       'Project name.
Dim ProjectStart As String      'Project start date.
```

```
Dim ProjectFinish As String     'Project finish date.
Dim Project As RepositoryObject 'Project repository object.
Dim OBJID_Project As Variant    'Object identifier for the Project class
                                    'definition object.
Dim TypeLib As ReposTypeLib     'GUID identifier for the SCS type
                                    'library.
```

The project information is stripped from the project object and assigned to the variables defined above. In this case, the project object contains information about the name of the project and its scheduled start and finished date.

```
ProjectName = ProjCol("Name")   'Retrieve the project information from
ProjectStart = ProjCol("Start") 'the project collection.
ProjectFinish = ProjCol("Finish")
```

In this particular case, information from the SCSapp is being written into Repository. In order to write information into Repository, Repository must first know how the information is to be structured. The 'Set TypeLib' statement identifies to the Repository the 'SCSTypeLib,' a predefined information model, that Repository is to use to structure the information as it writes it in.

```
'Access the Repository type library that represents the SCS type
'information model.
Set TypeLib = Root("IManageReposTypeLib").ReposTypelibs("SCSTypeLib")
```

Before creating the project object in Repository, a unique identifier must first be set aside that will be assigned to the project object when it is created. The following statement generates a globally unique object identifier to be assigned to the project object on its creation.

```
'Set the object identifier for the Project class definition object.
OBJID_Project = TypeLib.ReposTypeInfos("Project").ObjectID
```

The statement creates the actual project object.

```
'Create an instance of the Project class.
Set Project = Repos.CreateObject(OBJID_Project)
```

Once the project object has been created it remains to be referenced by the repository. Like most object-oriented systems, an object can only exist if there is a reference to it somewhere in the

system. This statement names the project object after the project's name and adds the project object to the 'SCS' collection. The SCS collection is a user-defined collection which exists on the Repository Root object.

```
'Name the instance after the project name and populate its start and
'finish properties.
Call Root("IReposRoot").SCS.Add(Project, ProjectName)
```

Once the project object has been added to the Repository, it can be populated with information. In this case, we expose the 'IProject' interface of the project object and assign the scheduled start and finish dates to the project object.

```
Project("IProject").ProjectStart = ProjectStart
Project("IProject").ProjectFinish = ProjectFinish
```

Once the project object has been created in the Repository, the Resource and Work Task Objects can be created. The following are the variable declaration statements required for the creation of a resource object in Repository.

```
Dim ResCol As Collection          'Resource information collection.
Dim ResourceName As String        'Resource Name.
Dim ResAvailableFrom As String    'Resource availability (from) date.
Dim ResAvailableTo As String      'Resource availability (to) date.
Dim Resource As RepositoryObject      'Resource Repository object.
Dim OBJID_Resource As Variant     'Object identifier for the Resource class
                                      'definition object.
Dim i As Long                     ' Counter.
```

The first task is to strip the resource object of its information and assign that information to the variables defined above.

```
'For each Resource, retrieve its information collection.
For i = 1 To ResCols.Count
  Set ResCol = New Collection
  Set ResCol = ResCols(i)
  ResourceName = ResCol("Name")
  ResAvailableFrom = ResCol("AvailableFrom")
  ResAvailableTo = ResCol("AvailableTo")
```

A unique object identifier is created. The Repository resource object is created and the object identifier is assigned to it.

```
'Set the object identifier for the Resource class definition object.
OBJID_Resource = TypeLib.ReposTypeInfos("Resource").ObjectID
'Create a new resource class instance.
Set Resource = Repos.CreateObject(OBJID_Resource)
```

Once the resource object is created it must also be referenced by the system. The information model to be used has already been identified (the 'Set TypeLib' statement seen above) so the resource object will understand its own structure and its relationship to other objects. In that information model a relationship exists between projects and resources – projects own resources. Therefore, the resource is attached to the 'Project owns Resources' collection on the Project interface (this appears as the first 'Resource' instance in the statement below).

```
'Name the new instance and add it to the Project Repository Object's
'"Resources" collection.
Call Project("IProject").Resources.Add(Resource, ResourceName)
```

Once the resource object is created, it can be populated with its availability information. The 'Next' statement closes the loop and the whole process repeats until all resource objects have been added.

```
    'Populate the Resource Object's properties.
    Resource("IResource").AvailableFrom = ResAvailableFrom
    Resource("IResource").AvailableTo = ResAvailableTo
Next
```

The process for adding Work Task objects to Repository is almost identical to the process for adding Resource objects that was just described and will not be described again. One difference exists though that is worthy of note: work tasks use resources and as such contain a collection of resources. In other words, resources may exist in two places – in a resources collection attached to the project object AND in a resources collection attached to the work task object.

When creating a work task object, it is necessary to determine which resource objects belong to that work task object and add it to the work task object's resource collection. Each work task maintains a list of resources it uses. This list is compared to the complete list of resources contained in the project object's resource collection.

```
'Find the existing Resource Repository Object in the Repository.
Set Resource = Project("IProject").Resources.Item(ResName)
```

Once the resource object has been identified, it is added to the work task object's resource collection so that it will exist in both places. This process is repeated until every resource that the work task uses has been identified and attached to the work task object's resource collection.

```
'Add it to the WorkTask Repository Object's "Resources" collection.
Call WorkTask("IWorkTask").Resources.Add(Resource, ResName)
```

Adding information to Repository does not actually write the information to the Repository database. To actually persist the information, the code must conclude with a Repository Transaction Committal statement. This statement will commit all changes and additions to the Repository database. The 'End Sub' statement ends the subroutine.

```
Repos.Transaction.Commit  'Commit the transaction.

End Sub
```