The IOP Approach to Enterprise Frameworks

Dr. Udo Nink, Stefan Schäfer

CronideSoft AG
Dachsgang 27
D-35428 Langgöns
(udo.nink | stefan.schaefer)@cronidesoft.com

Abstract: This paper introduces the Internet Operating Platform (IOP), an enterprise framework for large scale software development. In addition to obeying to important standards (UML, XML, Java) an enterprise framework has to fulfil three basic requirements. First of all, it has to be broad and needs an elaborate architecture complementing standards and technologies rather than purely connecting them. Therefore, IOP combines UML modeling, workflow specification, code generation, run-time configuration, and component architectures. Secondly, an enterprise framework allows most developers to concentrate on business leaving technical issues to a few specialists. Therefore, IOP abstracts from underlying technologies in the areas of front-ends (HTML, XML, Java), communication protocols (FTP, HTTP, JMS, RMI), distributed components (EJB, CORBA), and persistence (virtual memory, XML, SQL92, SQL:1999). All corresponding drivers are replaceable and can even coexist. Third, an enterprise framework has to provide micro solutions on both technical and business levels. Thus, IOP provides amongst others built-in services and components like session management on a technical level and content management on a business level.

1 Introduction

One major IT problem is the fast pace of changing technologies implying frequent changes of products, applications, market requirements, and education needs. A very promising approach to keeping pace with progressing technologies is to "think in **platforms**". In the automotive industry Volkswagen has been very successful with its realization of a platform for manufacturing and selling its car types as well as those of Audi, Seat, and Skoda (belonging to the same enterprise). As a side effect to the now unified technology and product stack solutions to other problem domains come along (uniform set of skills, uniform processes, and so on).

Naturally, software industry is making every effort to force IT platforms by definition of **standards**. In the past, single companies were able to establish de-facto standards due to the wide-spread use of their products. But today, very important as well as world-wide accepted de-jure standards emerge pushed by Internet technologies (HTTP, HTML, XHTML, XML, XSL, JAVA, Servlets, JSP, EJB, JMS [W3C, OMGa, Co01, Ro99]). Moreover, with UML [RJB99] and SQL [ISO99] we share international standards for software engineering and database languages, respectively.

In addition, the Internet forces application developers to "think in **services**". Such services (like registration services) embody high reuse potential compared to customer-specific business processes or fine-grained entities.

1.1 The Need for Enterprise Frameworks

Strategic software development has to address the above mentioned key factors. Approaches to implement such a strategy range from out-of-the-box solutions to custom development. From our observations in the areas of markets, projects, applications, and technologies (in **bold-face**, see list below) the answer lies in the middle and is named enterprise frameworks. Our approach to enterprise framework is IOP, the Internet Operating Platform [Cr02, Sc03].

Markets: Out-of-the-box solutions are adequate in near-perfect-fit cases, but fail for high customization effort. On the other hand, from-scratch-solutions lead to high overhead in choosing and implementing with the right set of products and technologies. Enterprise frameworks are suited when customization is expected to make up more than 30 per cent. The wheel gets invented only once providing a set of products and technologies that fit together and all applications on top save the effort.
Projects: Project management shall keep projects in-time and in-budget. Team members have to fit certain roles, be trained and be coached according to chosen tools and development processes. An enterprise framework already provides an overall architecture along which responsibilities, development activities, and roles are identified and aligned. One framework expert suffices to train and coach 10 to 20 team members.
Applications: Risk mitigation implies that technical problems are solved first. Consequently, development focus initially lies on technical issues and shifts to business issues with each iteration. Break-even is reached earlier with enterprise frameworks because of already proven technology stacks spanning user interfaces, workflow, communication, integration, and persistence as well as ready-to-use, thoroughly tested, partial solutions.
Technologies: The fast paced creation of standards, technologies, and product versions periodically "makes rookies out of experts again". Enterprise frameworks provide a more stable development environment buffering IT evolution to a certain degree. They provide a platform for choosing among different technologies for different subjects (like communication) in different contexts (like calling services). And technologies can be tested and compared by integration into the framework.

1.2 Roadmap

In the following we will introduce and discuss IOP – our implementation of an enterprise framework. Section 2 introduces IOP due to different views on its architecture. Afterwards, IOP objects and IOP components are detailed in section 3. Section 4 describes IOP interaction, IOP workflow, and their collaboration. Section 5 deals with modeling and code generation and how these are embedded in a phase plan for developing an example application. The relationship to other work is sketched in section 6. Section 7 concludes our work and gives an outlook to future efforts.

2 Architecture

We will start with IOP's design goals. Then, we discuss IOP's architecture from different perspectives: building blocks, layering, systems, and topology. The building block view starts with a concise list of main concepts and micro frameworks. Then, the relationships between these blocks are discussed in the layer view. Afterwards, the system view gives an overview of grouping functionality into subsystems. The topology view describes the basics for installation and configuration. Finally, we compare different architectures.

2.1 Design Goals

IOP has an object-oriented distributed component architecture including methods and tools for the software development life-cycle. And it has a strong focus on standards. Three ubiquitous standards build IOP's foundation: UML, XML, and Java/J2EE:

- **UML** (Unified Modeling Language [RJB99]) is the base of visual modeling.
- **XML** (eXtendend Markup Language [Co01]) is the base of data exchange and configuration. Furthermore, it is used as alternative persistence model.
- **Java** is the programming language of choice; important **J2EE** packages [Ro99] are integrated with IOP. But, while J2EE is merely a huge unsorted box of APIs IOP delivers the glue putting the pieces together via a sophisticated architecture.

All other supported standards and technologies are replaceable throughout the framework from front-end to database:

- **Front-end**: Coupling of arbitrary front-end technologies in multi-channel fashion via IOP Interaction. Implementation proves include Java Swing, HTML, DHTML, and WML.Markup generation is supported using JSP, Servlets, and XML/XSLT.
- **Workflow**: Process definition via WPDL (Workflow Process Definition Language [WfMC]) or via UML activity diagrams (planned). IOP provides two levels of workflows: simple workflows executed on very fast core engines and complex workflows executed on dedicated workflow components on top of core engines.
- **Communication**: Different protocols encapsulated as IOP devices like HTTP, RMI, FTP, POP3, SMTP, and JMS. Such devices can be reused via drivers for component communication and for the IOP Virtual File System (VFS).
- **Component architecture**: Support for CORBA and EJB. IOP provides a common base for developing components based on CORBA, EJB, or internal IOP concepts.
- **Persistence**: Configurable persistence managers for virtual memory (VM), ORDBMS (SQL:1999), RDBMS (SQL92), and file system (XML) using persistence mappings (object-to-relational, object-to-object-relational, [Sc03, Ru01, Am99]).

IOP does extensive code generation. From the UML model IOP generates configuration information, Java code, and SQL code for the handling of objects, object references, object collections, and more. In addition, workflows are compiled to the core workflow engine. See section 5 on “Modeling and Code Generation” for a complete example.

2.2 Building Block View

IOP is made up of several building blocks which are roughly described in the following:

IOP Objects: Objects (more precise: business objects or user-defined object types) are classes in the UML model with stereotype "IOPBO" for IOP Business Object. They represent the smallest building block of IOP. Visually designed object models including direct relationships and inheritance are supported. These models define the static structure of object graphs that hold application data and provide for local-scope functionality. Objects have location-transparent identity, are type-safe, and can be versioned and referenced. Object graphs can declaratively be copied and be transformed between different representations (virtual memory, XML, SQL92, and SQL:1999).
IOP Components: IOP Components aggregate and manage IOP Business Objects. They provide for vertical business logic combined into vertical services. Components are the granule for transparent distribution and addressing as well as for integration of third-party services and applications. Moreover, they are key to optimization of performance and scalability. Finally, components encapsulate underlying component architectures and communication technologies. More than a dozen of ready-to-use or ready-to-adapt components are given ranging from id handling to content management.
IOP Design Repository: The IOP Design Repository holds all static design information extracted from the UML models by the IOP Compilers; its corresponding component is named "IOP Design Component". This information is enriched by mapping information for Java and SQL. Currently supported partial UML models are the class model and the component model. Dynamic design information is supported via workflows (see Workflow Framework below).
IOP Run-time Repository: The run-time repository is driven by the IOP Configuration Component. It holds information corresponding to topology (hardware nodes, software nodes, module nodes), configuration of components w. r. t. choices of technologies (drivers, devices, persistence, virtual file system), and initialization data ranging from passing of user data to specifications of load balancing and fallback.
IOP Service Framework: The IOP Service Framework is a collection of so-called micro frameworks that encapsulate partial and mostly technical solutions. Micro frameworks provide reusable services via stable interfaces. Currently, more than a dozen services are implemented like for localization, logging, or the virtual file system. A very important role is played by the device/driver concept which is used consistently throughout the framework. Devices encapsulate low-level APIs for close to ten different protocols like ftp and http. Drivers like a file system driver can be implemented on top of such devices. The drivers, in turn, can then be used for configuring instantiated services like file systems for components or software nodes. Since IOP supports parallel usage of different implementations at the same time you can easily construct file systems hiding different protocols behind simple folders much like in Unix operating systems.
IOP Persistence Framework: The IOP Persistence Framework provides for persistent storage of Java objects. It combines the concepts of SUN JDO [Ru01] and well-known concepts for object-relational mappings [Am99]. Four persistence mappings are available in IOP: IOP Virtual Memory Persistence, IOP XML Persistence, IOP SQL92 Persistence, and IOP SQL:1999 Persistence [Sc03]. Needed mapping information is extracted from the UML models and stored with each object type in the IOP Design

Repository and in the IOP Object Type System. Again, components can choose persistence mappings by configuration. A very nice spin-off of this approach is, e. g., that exporting database data (from a database wrapped by a database-driven component) is reduced to copying the data to a component driven by IOP XML Persistence!

IOP Interaction Framework: The IOP Interaction Framework defines the access point to an IOP system for the outside world. It is based on a service handler architecture and is responsible for converting requests to and responses from any IOP system. Encapsulated communication protocols (HTTP, JMS, RMI) and exchange formats (HTML, WML, XML, Java objects, messages) are implemented for various servers (like application servers). Implementation choices are, again, configurable.

IOP Workflow Framework: The IOP Workflow Framework is based on the reference architecture of the Workflow Management Coalition (WfMC, [WfMC]) and the corresponding OMG recommendations [OMGb]. IOP thus supports workflow specification via WPDL. Workflows (or processes) are configurable in terms of distributed execution, auditing, and statistics. We distinguish between workflow definition and workflow runtime for performance-improved representations. Workflow participants are mapped to an organizational model thus leveraging from existing organization data in business applications. Robustness and scalability are given by transaction awareness and disconnected session management. Since workflows can span many scopes and timescales (business workflow, user interaction workflow, technical workflow) we provide for a very fast core workflow engine as basis for more specialized engines (e. g. for user interaction).

IOP Integration Framework: The IOP Integration Framework collects concepts, implementations, and workflows for third-party systems integration. Integration components act as clients of such systems. Due to resulting intersections between integration and interaction, both micro frameworks share common code: low-level drivers for communication protocols, exchange formats, and implied transformations. Moreover, integration can reuse interaction by, e. g., registering an interaction listener on a message-oriented integration hub. Finally, integration components can also act as persistence managers thus providing access to integration data via ordinary business objects.

IOP Code Generation Framework: The IOP Code Generation Framework provides a set of compilers (including scanners, parsers, analyzers, and writers) supporting code generation for modeled design information. These compilers are heavily used for application development as well as for development of large portions of the framework itself. This self-reproducing feature guarantees continuous testing of IOP. All design information is stored in the IOP Design Component and the IOP Object Type System. Supported source languages are UML and WPDL and corresponding target languages are Java, SQL92, SQL:1999, XML, and IOP Workflow Format. Each generated business object for Java obeys to a set of inner interfaces implemented by corresponding classes for its entity (storing data), behavior (operations), collections (array, map), referencing (object linking), and persistence mappings. Moreover, SQL code is generated for definitions of types and tables. Additionally, DTD/XMLSchema code is generated structuring XML representation of business objects. Finally, workflows defined via WPDL are translated into IOP Workflow Format that can be handled by the IOP Core Workflow Engine.

2.3 Layer View

Let us now put the building blocks together. Figure 1 shows the layer view on IOP's architecture. The layers are: presentation layer (on top), application layer, component layer, and persistence layer. Starting left of the component layer you will find a UML

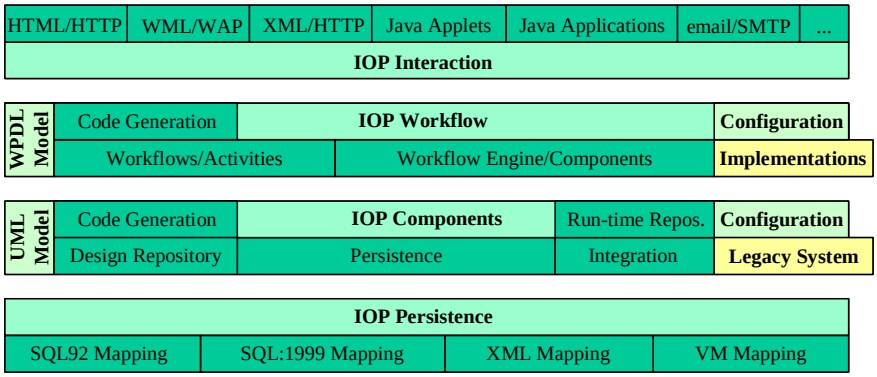

figure 1: Layer View on IOP Architecture

model containing the system design. This model is parsed and written to the IOP Design Repository. During code generation business components and internal components are generated. These can be configured to reuse existing micro frameworks like integration or persistence. The configuration (see right side) is written into the run-time repository. In case of integration a component wraps an outside system (e. g., legacy system). If a component configures a persistence manager it can choose between mappings for virtual memory persistence, xml persistence (xml files in file system), or SQL persistence (see persistence layer at the bottom).

In the application layer (2nd from top) workflows represent business processes defined in WPDL. Again, the code generator delivers internal representations. These get executed by workflow engines/components. Each such workflow consists of an activity network where activities wrap implementations. Such implementations can be external applications or IOP Executables which, in turn, coordinate calls to business components and prepare data sets for exchange with the front-end.

The presentation layer on top is served by IOP Interaction which is responsible for translating and dispatching requests and for delivering channel-specific responses. If user interaction is needed in a workflow then you define the corresponding views to be resolved by IOP Interaction as attributes to the corresponding activities. Such a view might be given by an ordinary html page uploaded into the IOP Content Component.

2.4 System View

The system view of the IOP architecture describes subsystems, that is, sets of interfaces (services) encapsulated for software nodes. The idea is to have services running in separate processes. While some subsystems are inherently needed for an IOP system others can be reused or adapted for application programming. Similar to operating systems different run levels allow for setting an IOP system to certain maintenance modes. Each (higher) run level adds functionality (e. g., run level 3 supports component startup/shutdown). Since IOP is built in Java, processes correspond to instantiated Java Virtual Machines (JVM). For the sake of performance and scalability, all needed IOP resources for JVMs are managed by the IOP Resource Manager. Here is a list of subsystems:

IOP Boot System: Starting from a single small property file various boot strap loaders are coordinated to start up or shut down the IOP kernel.
IOP Kernel System: All subsystems and any application system running on IOP are managed by the kernel system. It is responsible for switching between run levels.
IOP Configuration System: Manages IOP installations, that is, their topologies. These include hardware and software nodes, component deployment, driver choices, queries, access policies, devices, listeners, persistence, and caching.
IOP Session System: Sessions store run-time or state information during interaction with the IOP system (often user-specific information). A session system manages all sessions for a software node.
IOP Logger System: Logging is important for detecting errors and misuses. The logger system provides logging services for quickly storing log messages at certain software nodes. Log messages can be leveled (by severity) and categorized (by category and subcategory). Developers can then use output filters to quickly search for error causes when compiling or running IOP.
IOP Localization System: Localization is used in multi-language applications. It is message-based and maps message numbers to locale-specific messages. Message number ranges are also supported. Developers do only use message numbers when coding while associated message texts are defined and resolved in one place.
IOP Driver System: Many APIs (e. g., file system) and protocols (like ftp, http) can be plugged into IOP via device drivers. A driver system manages all driver instances for a software node. Each software node can thus be configured to provide certain drivers. Consequently, software nodes can be dedicated to certain APIs and protocols.
IOP Component System: Components are running in component containers. Management of component instances at run time is in the responsibility of the component system. Software nodes can be configured to provide only certain components. While drivers map APIs to protocols, components implement low-level business logic reusing drivers that abstract from technical protocols.
IOP Workflow System: Very fast core workflow engines associated, again, to software nodes execute defined workflows. Workflows can be complex (inter-department) or simple (local scope, no user interaction), Simple workflows can be run on dedicated workflow-sensitive components to improve performance or do functional enrichment.
IOP Object Location System: Objects have a logical location which is part of their OIDs. Locations map to storage managers (components in most cases). The object location system resolves locations given by OIDs (or object references).

IOP Object Type System: Only strongly typed software can be made reliable and efficient with acceptable effort. Thus, IOP forces strong typing (each object belongs to an object type). All available object types (IOP-specific and application-specific) are managed by the object type system. These are represented by meta objects for object types, components, object members, and corresponding persistence mappings.
IOP Transaction System: Critical business processes have to be transactional. Thus, the transaction system is responsible for encapsulating transaction systems like transaction monitors. IOP is based on JTS and JTA [Ro99].
IOP Statistics System: Similar to logging it is important to collect business information during system interactions. Again, storing of such information must be fast while analysis itself can be deferred and be done asynchronously. The statistics system provides services for storage and analysis.
IOP Dispatcher System: For the sake of scalability the dispatcher system delegates incoming requests to subsystems, workflow engines, and components which then become responsible for serving corresponding requests.
IOP Command System: The command system realizes a command shell to an IOP system. Command shells are well-suited for testing during development, for administration purposes, and for simple batch jobs using command scripting.

2.5 Topology View

Figure 2 shows the first part of the topology view on IOP concentrating on installation configuration as it is defined in our UML class diagrams. The configuration information is collected under an object of type IOPInstallation with stereotype IOPBO (meaning IOP Business Object). An installation pools host nodes, file systems, loggers, and components. Host nodes represent hardware (computers, processors) running software nodes (processes). Associated file systems are used for file-based input/output (e. g., a content component might import contents from a file system). Loggers are needed for storing log messages – each software node must have exactly one logger. The components in the pool can be wrapped by component instances (now see figure 3) which are associated with software nodes.

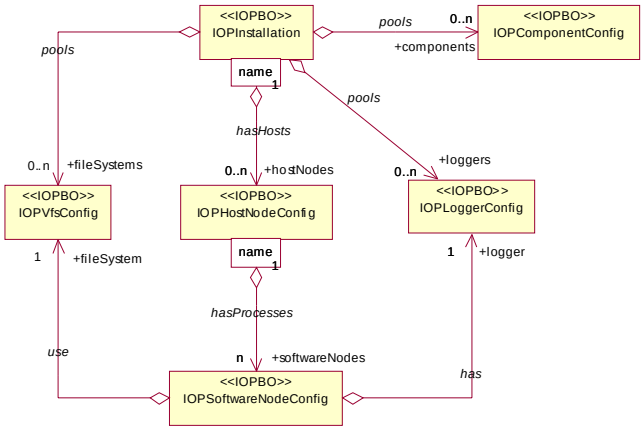

figure 2: Topology View (part 1) on IOP

The components have to be provided by component containers (like Oracle9i AS, Bea WebLogic, and Apache Tomcat). Component containers provide component servants and component drivers. Component servants only contain implemented business logic. Technology-specific code like home classes or remote classes are generically provided by IOP and, thus, need not be hand-coded by component programmers.

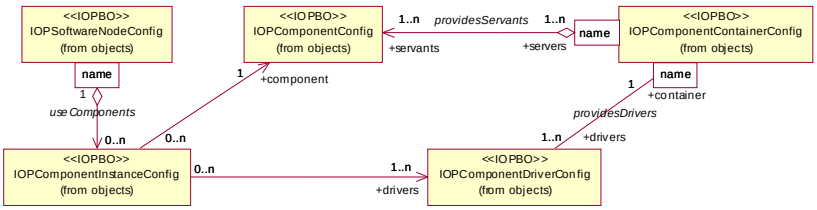

figure 3: Topology View (part 2) on IOP

Component drivers allow to communicate with components using different protocols. Redundancy and fallback (not shown here) are supported by associating redundant and fallback drivers. A software node uses components by defining component instances. Component instances link chosen component servants with chosen component drivers.

2.7 Comparison of Architectures

Figure 4 shows three different kinds of layered application architectures on the top and two example architectures at the bottom.

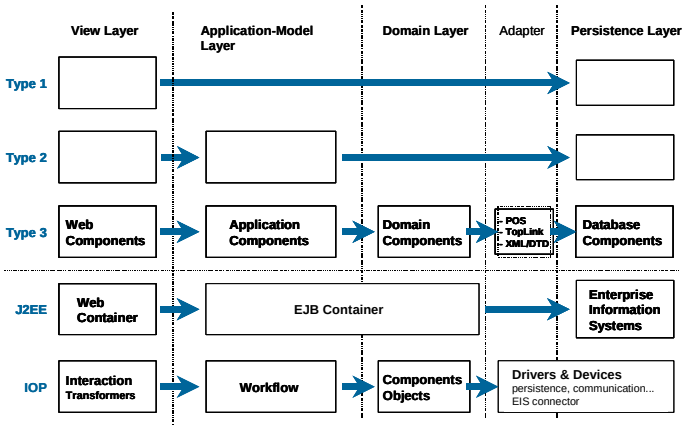

figure 4: Layered Architectures

Type 1 leads to monolithic clients mixing domain logic, application logic, and GUI logic. Type 2 introduces application components allowing for thinner clients and for reusable application logic. Type 3 enriches this model by introducing domain components collecting common application logic across different applications. Moreover, a persistence adapter layer is shown due to the technical necessity to bridge the gap from programming languages like Java to databases (impedance mismatch).

The two examples at the bottom of the figure, J2EE and IOP, are both type 3 architectures. J2EE introduces the notion of containers:

- Requests are served by web containers supporting Servlets and JSP having access to the J2EE APIs (JMS, JAAS, JTA, Java Mail, JAF, JAXP, JDBC, and connectors).
- For the application-model layer as well as for the domain layer J2EE provides EJB containers where EJBs, again, have access to the mentioned J2EE APIs.
- Access to databases (or, in general, to enterprise information systems) is typically realized using connectors, JDBC, or JMS from within EJBs of the domain layer.

IOP, first of all, allows to implement the same technical architecture as J2EE. But, in contrast, it also delivers a more process-oriented architecture as shown in the figure.

- In the left IOP Interaction serves the view layer in order to encapsulate all UI-specific communication (interaction) to applications. In addition to J2EE markup frontends IOP also supports Java Swing. In addition to http as communication protocol IOP also supports technologies like wap and smtp allowing to serve requests via mobile devices and email, respectively. A transformer approach allows to plug in channel-specific translators for requests and responses.
- For the application-model layer IOP provides IOP Workflow. Requests are mapped to workflows representing application logic and delivering responses as well as model data for driving the frontend using the MVC pattern. If by-pass is needed then requests can also be mapped directly to activities and components.
- For the domain layer IOP provides IOP Components. Implementation technologies like EJB or CORBA are hidden and can be chosen by configuration of drivers and devices. Exactly the same concept is used for connecting to databases and third-party systems.

3 Objects and Components

Section 2.2 already introduced the notions of IOP Objects and IOP Components giving a short overview over both concepts. Now, the concepts shall be detailed.

3.1 Object Types

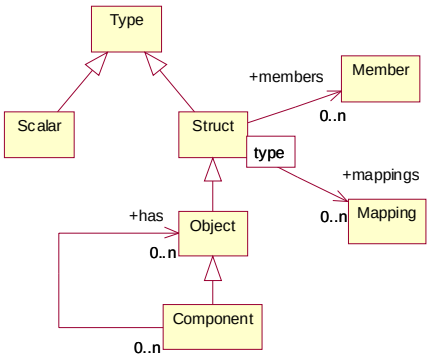

figure 5: IOP Type System Model

IOP delivers an enhanced type system having two main advantages over the Java reflection API: it is more expressive w. r. t. versions, multiplicity, containment, and persistence mapping; and it performs better (optimized member access via indices). The UML diagram in figure 5 shows our type model. The root class named “Type” is abstract and only gathers common concepts for

all kinds of types. Subclasses are “Scalar” and “Struct”. IOP provides close to 20 scalar types (like blob, boolean, byte, char, and so on). These are mapped to Java primitive types and can be used as such in a type-safe manner. When modeling, you can also choose object scalars that have been introduced to support null values. “Struct” is used for record-like structures and, thus, has members and member mappings. “Object” inherits from “Struct”. In addition, objects are identifiable, that is, have an object identification, OID in short. OIDs are globally unique identifiers and allow for location of objects and building relationships (using references) independent of their physical storage location. An OID in IOP is made up of five values: location number, type number, type version, instance number, and instance version. In addition, many operations are predefined which all IOP Objects get for free.

3.2 Object Interfaces

Modeled object types are input to the IOP code generators which deliver a set of Java classes obeying to a set of interfaces given in figure 5. On top you find the interface “Entity” collecting member setters and getters. The interface “Behavior” collects modeled and implemented operations. All other interfaces act as clients against behavior interfaces in order to use objects in different contexts. The interface “Reference” is a proxy - all object operations can be called using either the object itself or a reference to it. Dereferencing occurs transparently regardless of object locations. Furthermore, collection support is given by implemented interfaces “Collection” and “List” both physically backed up by “Array”. The interface “Map” allows for keyed access to its entries. For set-oriented operations IOP provides a universal graph interface. On top of it you get intra-object-graph navigation via object cursors and fast object graph transformation between virtual memory representation, maps, XML, and SQL.

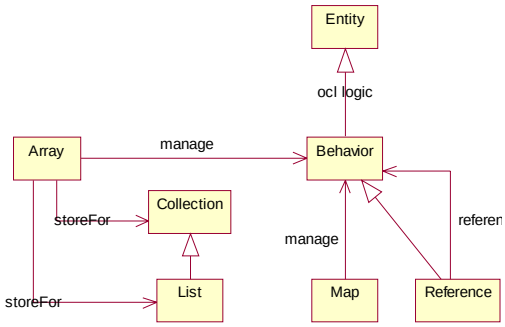

figure 5: Object Interfaces in IOP

3.3 Component Model

In the section on architecture we have discussed commonalties and differences of IOP and J2EE. Both have component models. The component model of J2EE, EJB, will be compared to IOP Components. In addition, we will give an idea on the differences to web services and what role web services can play in an IOP environment.

Since IOP allows to use EJB as technology driver for IOP Components, you can leverage from existing EJB concepts. In addition, IOP has the following main improvements:

- IOP supports EJB as well as CORBA or pure IOP Components.
- The resource management of EJB concentrates on threads, socket connections, database connections, and, of course, components. IOP also has a resource management. It knows the following resources: component containers (like tomcat for servlets or jboss for EJBs), component instances, workflow engines, and any drivers (persistence mappings, communication protocols, virtual file systems).
- IOP Components have a built-in core workflow engine that allows for execution of simple workflows. Thus, workflows can be used to chain component operations inside a component to form new operations without Java programming.
- EJB relies on RMI, RMI-IIOP, and JMS for distributed object communication. IOP additionally allows arbitrary protocols as long as drivers exist (e. g., http).
- EJBs can make use of any Java class. On the other hand, there is no global data model available across EJBs. IOP additionally supports UML class models. These can be partitioned into partial models. Each partial model can be associated to a component and, thus, become its schema. The code inside an EJB may rely only on that schema meaning that Java code as well as database queries are based on this schema. Programmers do not need to write mapping code, but rather choose persistence drivers. Of course, databases can also be directly accessed via JDBC.

Taking a look at web services we first talk about what web services are meant for, what they are, and what they are not:

- Web services are meant for communication between applications across networks and firewalls. Nevertheless, it is possible to use them for inter-component communication inside applications, too.
- Web services are useful for publication and invocation of services. They do not help you w. r. t. service complexity or assembling of services.
- Web services can be viewed as component model. In fact, they provide distributed object communication mainly by specifying component interfaces and XML messaging. But, they do not provide a programming model for implementing services or components like EJB or IOP do.

Web services are not yet part of IOP, but integration is straight forward. We view web services as yet another technology that can be configured for certain responsibilities:

- IOP can use web services via drivers for inter component communication and calls from activities to components.
- Some IOP concepts are well suited to be published as web services by generating WSDL and integrating SOAP into IOP Interaction. Candidate concepts are workflows, single activities, components, and component methods.
- IOP will not provide an UDDI implementation, since UDDI is used for global registering and finding of web services. It suffices to support access to UDDI directories.

3.4 Components

More than a dozen components are already implemented. In some cases existing IOP services are only complemented by components because of the basic need for persistent storage (left out in the list below). While some components are rather technically motivated others have been built for application development (marked "business" below).

IOP Design Component: The design component manages all elements extracted from UML modeling by model file parsers. It conforms to UML 1.3 (soon UML 1.4) with regards to class models and activity models and enriches them by needed mapping information. The so parsed models are input for the code generators for Java and SQL.
IOP ID Component: The id component is responsible for creating new OIDs with which newly inserted objects shall be stored.
IOP Workflow Definition Component: Workflow models specified via WPDL are compiled by the workflow compiler. The resulting process definitions are managed in the workflow definition component. A workflow model basically contains workflows, activities, transitions, applications, participants, and workflow relevant data.
IOP Workflow Instance Component: When instantiating a workflow its definition is first fetched from the workflow definition component. The instantiated and configured process is then stored and thus available for execution by the workflow engine.
IOP Content Run-Time Component: This component is used to optimize content management information for run-time presentation. On one hand, it provides optimized physical contents for run-time access. On the other hand, it can also provide multiple physical contents for each logical content to support multi-channeling.
IOP Content Management Component (business): The content management component supports multi-provider content sites. A site can be bulk-loaded without the need to manually specify each single content. It is basically structured into sub-sites, resources, frames, and elements. In addition to managing arbitrary files the component is also used for managing markup pages for building application front-ends.
IOP Batch Component (business): This component is not explicitly programmed. It is rather a reminder that its functionality is already given by IOP workflows. That is, workflows can be time-sensitive, and a dedicated workflow engine can be configured to control workflow execution using a clock. Consequently, periodic deliveries, nightly backups, weekend reports, and alike are built using time-sensitive workflows.
IOP Actor Component (business): All actions are performed by actors (e. g., participants perform workflows). Thus, the actor component allows for management of actors having accounts and passwords. Actors are also the basis for access restrictions managed by the access component.
IOP Access Component (business): The access component manages access control lists giving accessors access to accessibles. In most cases actors will take the role of accessors and some products will take the role of accessibles. For the sake of flexibility the access component itself does not pose any restrictions here, so that arbitrary objects may take the role of accessors or accessibles.
IOP Organization Component (business): The organization component adds roles and organizational units to actors. According to directory services or participant mappings it allows to represent complete organization structures where organizational units provide roles and manage actors taking roles.

4 Interaction and Workflow

Interaction is responsible for, basically, converting requests to and responses from an IOP system. An incoming request is translated into an IOP Message. This message is then sent to the dispatcher. The dispatcher identifies the responsible workflow that shall serve the request. The workflow (as well as implied activities, executables, and component operations) is then run until the next point of interaction. The result is, in turn, sent back as a message to the dispatcher. The dispatcher identifies the corresponding request and forwards the message to the interaction layer. The interaction layer finally translates the result message into a response leaving the system. The workflow to be executed is wrapped by a technical workflow allowing for pre- and post-processing. Thus, interaction is easily customizable by adapted technical workflows or simply by replacing the therein called executables.

Workflows are defined using WPDL. The definition spans complete workflow models including workflows, activities, transitions, applications, participants, and workflow relevant data. Unfortunately, many things have not been taken into account when WPDL was defined by WfMC – it is an open issue where to put configuration information for workflow models. For instance, you somewhere have to specify communication channels. As a result, workflow system vendors heavily use the so-called extended attributes (freely definable lists of name/value pairs) in order to backpack configuration information to the workflow definitions. Consequently, interoperability between workflow systems gets a lot harder.

During workflow execution instantiated workflow definitions are kept in the workflow instance component. An instance holds run-time information for exactly one associated run of a workflow. This run-time information is fed into the core workflow engine. The workflow system supports transactional workflows and disconnected sessions in order to make workflows reliable and to handle interruptions in case of manual activities.
Participants in a workflow model have to be mapped to persons or systems (actors in our case). Again, the WfMC leaves it to you, but at least recommends the use of organizational models. IOP therefore supports mapping of participants to actors, roles, and organizational units.
Participants might be involved as performers of several workflows and activities. Thus, work lists are supported that collect all work items (workflows and activities) a participant is responsible for. Such work lists are the base for pull or push approaches where participants or the system decide when to work on which item.

5 Modeling and Code Generation

IOP supports full project life cycles and corresponding development processes (methodologies). But, instead of defining yet another development process we rather define steps that you have to undertake during development (what is done by whom when and why). Since theory of development processes is a huge area we decided to present only an example in this paper. Our example will be given in the order sketched by figure 6. We

will first sketch some requirements and then discuss partial models that should result from analysis and design (see complex activity “modeling” in the activity diagram above). The next complex activity “code generation” introduces code generation support of IOP. The final complex activity “coding and configuration” hints on necessary hand-coding and system configuration.

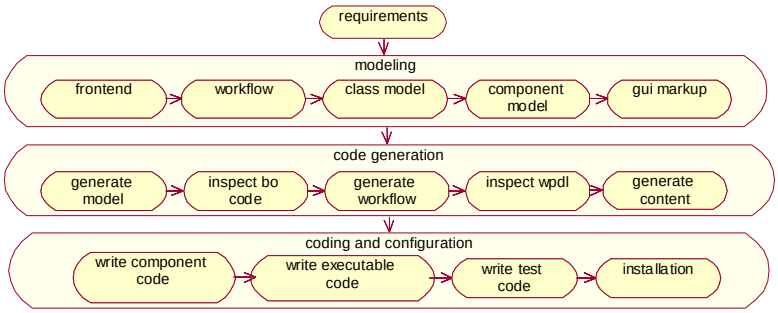

figure 6: Recipe for Example Application

5.1 Requirements

Our example application is named “WikiCms”. It shall combine automatic linking features and ease of use of the wiki brainstorming tool with content management features. Following are the requirements (incomplete, but sufficient for our example):

- web front-end based on html
- complete import of existing file systems
- automatic content linking based on file system folders and subfolders

5.2 Partial Models

Let us start with a front-end page for importing a file system (see figure 7). It is a standard html page where you enter the source path of an import folder and a target mount point.

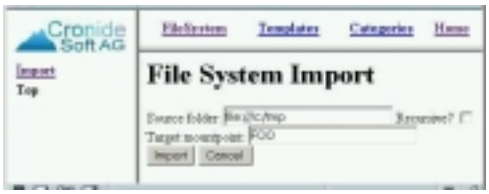

figure 7: Import Page

The corresponding workflow has four activities (see figure 8). Activity “Import_Start” imports the source folder and instantiates meta nodes in main memory. Then it (xor-) splits to either activity “Import_Save” (saving the object graph in the component) or “Import_Error” (analyzing the error cause) depending on the workflow relevant data named “successful” of type boolean. Both

activities (xor-) join again into activity “Import_End” that triggers the next front-end page to show up (see exit-action). Each activity is defined as “IMPLEMENTATION” of type “Tool” (see do-actions).

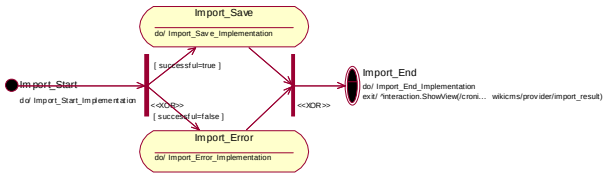

figure 8: Activity Diagram

The class model contains a class diagram on meta directory objects (see figure 9). Basically, we need the business objects “Meta Node” (folders and files), “Meta Directory” (subclass for folders), and “Meta File” (subclass for files). The objects are managed by the Meta Directory Component (component model not shown).

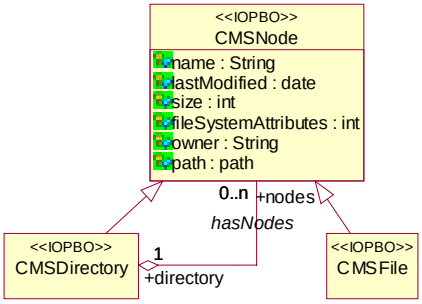

figure 9: Class Diagram

The page “top.htm” (see code below) is the start page for navigating through the imported file system. It is specified in html plus the IOP tag library (close to JSP). First, you initialize the model part of the model-view-controller pattern (MVC) by specifying a business object (“iop:useBean”) to use for data exchange between the activity (the controller part) associated to the page (the view part). Then, you specify access paths (“Directory.ObjectId”). A path can, e. g., be used for linking an activity to show up a node info page (see “action” inside “iop:a”) to the name of a folder (see “iop:value-of”). The “iop:for-each”-tag then loops through the collection of nodes under a directory.

```
<!--top.htm-->
...
<iop:useBean id="Directory"
  class="cronides...CMSDirectory" >
  ...

  <iop:input type="hidden"
    name="Directory.ObjectId"
    value="Directory.ObjectId"/>

  <iop:a href="action=shownodeinfo&amp;
    objectId={Directory.ObjectId}">
    <iop:value-of value="Directory.Name"/>
  </iop:a>
  ...

  <iop:for-each select="Directory.Nodes"
    alias="nodes" >
    ...
  </iop:for-each>
...
```

5.3 Code Generation

First, one compiles the UML model. The result is a nested folder or package structure containing generated Java sources (let us skip generation of XML and SQL). Furthermore, object type system and design component are updated. Now, we inspect the gener-

ated Java code. The example shows part of “CMSFile.java”. Amongst other snippets a package statement, an import section, the class definition (see “class”), a serial number (for mapping design entries to Java code snippets), constructors, and inner interfaces have been generated. All other generated classes (not shown here) for the business object deliver default implementations for the inner interfaces.

```
package ...component.metadirectory;
import  ...

public final class CMSFile
  extends IOPObject {
  static final long serialVersionUID = 859...;
  ...
//constructors
//inner interfaces for entity, behavior, ...
  ...
  public interface Behavior
    extends Entity, CMSNode.Behavior{}
  ...
}
```

The next step is to compile the definitions. Workflows can also be exported to and imported from WPDL and WPDL/XML (our XML-version of WPDL, upcoming XPDL proposal will be evaluated). The following code is an excerpt from the workflow model. You can see the definition of activity “Import_Start” which is of type IMPLEMENTATION and wraps the tool “Start_Implementation”. This tool is defined in the application section and links to a Java class via “ToolName”. You can also see that the activity defines an xor-split thus restricting the mentioned transitions. In case your input to the html page is correct, the workflow engine will decide to move on to the next activity “Import_Save” (or to activity “Import_Error” otherwise).

```
...
 <WorkflowProcessDefinition
   Id="WikiCMS.Import"
   Name="import" Created="2002-02-06">
   <Activity Id="Import_Start"
             Label="Import Start Node">
     <ActivityKind Type="IMPLEMENTATION">
       <ActivityImplementation>
         <GenericTool Tool=
           "Start_Implementation"/>
       </ActivityImplementation>
     </ActivityKind>
     <TransitionRestriction>
       <SplitCharacterisation Type="XOR"
         Transitions=
           "T_Start_Save T_Start_Error"/>
     </TransitionRestriction>
   </Activity>
   ...
   <Application Id="Start_Implementation"
     Label="Start Activity Implementa..."
     ToolName=
       "...CMSWfImportTraverseCode"/>
   ...
 </WorkflowProcessDefinition>
```

Next, you compile the content, that is the markup pages that make up the front-end of the application. Since IOP supports bulk-content upload your input can be a complete (web) site spanning all pages including supported file types like gif. The content compiler builds a content management structure describing the content from a logical perspective (sites, sub-sites, resources, elements, and so on). Then, it constructs a far more efficient run-time representation (compiled content) used by optimized content viewers. Content run-time also supports mapping of logical content to many physical contents by mime-type. At activity run time physical contents will be fetched according to specified target formats. In case of dynamic content the mentioned viewers assemble static and dynamic content snippets. Dynamic content snippets contain executable code for identifying and inserting business objects and their attributes.

5.4 Coding and Configuration

In order to complete the implementation stack you first implement designed components. Freed from technology issues corresponding to communication and storage you only concentrate on the low-level business logic. E. g., the meta directory component implements an operation named “search NodesByPath()” (see code). After initialization of local variables (like “query” which is in fact configured outside the component and now only identified by a name) query parameters are set (see “setPlaceHolderValue”) and the query is executed (see “execute”). The result set is then copied into a node list (see while-loop) which is returned to the caller.

```
//CMSMetaDirectoryComponent
public CMSMDNode.List searchNodesByPath
  ( String searchString )
  throws IOPComponentException
{
  ...
  IOPQuery query =
    getQuery("findNodesByPath");
  ...
  try {
    if(query != null) {
      query.setPlaceHolderValue
        (1,new,IOPPath(searchString));
      rs = query.execute( ap );
      while ( rs.hasNext() ) {
        node = (CMSMDNode.Entity)
          rs.nextObject();
        list.add(node);
      }
      rs.close();
    }
  }
  catch( IOPException e0 ) {...}
  return list;
}
```

Next, you implement executables (called by activities). The following example belongs to the executable “CMSShowFolder”. There is one public method named “execute()” (see code below). Access to components is prepared by calls to “getComponent()”. The page to be used for displaying folders is specified at the activity (using, e. g., the attribute View = ”/wikicms/top.htm”). The folder that shall be used to dynamically fill that page is simply handed over as an OID (see “objectId” and “getNode” in the code below and remember the code in the html page). The OID is read from the incoming request (user clicked a folder link before). Finally, you set the fetched node as model for the view.

```
//CMSShowFolder
public void execute()
  throws IOPWfException {
  ...
  mdc = (CMSMetaDirectoryComponent)
    getComponent( "metadirectory" );
  ...
  String oidString =
    getStringParameter("objectId");
  IOPObject.Id oid =
    IOPObjectId.valueOf(oidString);
  ...
  CMSMDNode.Entity node =
    mdc.getNode(oid);
  ...
  addViewModel("Directory", node);
  ...
}
```

To save space, we skip the testing code for unit testing, feature testing, and benchmark testing. And, we skip the installation file that contains the complete configuration of the IOP application. Basically, the structures defined earlier when discussing the topology view on IOP's architecture are instantiated as an XML file.

6 Relationship to Other Work

Based on [FHB02] we discuss some criteria for enterprise frameworks. A brief comparison of Abaxx eBusiness Suite, Interactive Objects ArcStyler, and CronideSoft IOP already shows the range in functionality of enterprise frameworks (see table below).

	Abaxx	**ArcStyler**	**IOP**
foundation	J2EE/EJB	UML, J2EE/EJB	UML, Java, XML
central aspect	assembling of multi-channel process portals	applicaton-server-specific code generation	code generation and technology encapsulation
method	-	Convergent Architecture	any method; tasks are predefined
focus of functionality	personalised content delivery, CRM	architectural IDE based on MDA	infrastructure bridging business and technology
gui support	+	--	++
protocols	+	--	++
component model	EJB	EJB, Web Services	IOP Components, EJB, CORBA
persistence mappings	--	--	++
workflow	proprietary	UML state charts	WPDL, WPDL-XML
# aspects	2 (caching , personalization)	1 (security patterns)	8 (caching, localization, logging, object type system…)
extensibility	-	+	++
# business themes	7 (access control, content management, data extr…)	0	5 (access control, content management, …)

Abaxx concentrates on ease-of-use providing a lot of tools and predefined business themes for building personalized portals. The suite fully depends on EJB, does not provide for persistence mappings, and is rather restricted in terms of extensibility (by assembling of workflows). ArcStyler comes out to be a very sophisticated IDE. It is a specialist in generating code for different EJB containers and heavily concentrates on a methodical approach based on Model-Driven Architecture. On the other side it does not provide direct support for gui, protocols, persistence mappings, nor business themes.

7 Conclusions

This paper has introduced IOP, the Internet Operating Platform. IOP is a high-end enterprise framework combining a large number of concepts, standards, implementations, and products to a synergetic whole. It is based on three major standards: UML for modeling, XML for data exchange and configuration, and Java as programming language. All other supported technologies are encapsulated for dynamic replacement or even coexistence via configuration in the areas of front-end (HTML, XML, Java), workflow (WPDL,

UML), communication (FTP, HTTP, JMS, RMI), component architecture (EJB, CORBA), and persistence (virtual memory, XML, SQL92, SQL:1999).

IOP yields the following benefits:

- *Productivity* is increased by encapsulating protocols and technologies, by visual modeling, extensive code generation, and the architecture-driven approach.
- *Quality* is increased by development process support, extensive use of design patterns, self-reproducing capabilities, and continuous self-testing due to the appliance of code generation for framework development itself.
- *Extensibility* is increased by the concepts of IOP Interaction, IOP Workflow, IOP Objects, IOP Components, and device/drivers for plugging in communication protocols, persistence mappings, component architectures, and external systems.
- *Flexibility* is increased by sticking to common open standards and by configurable interchange of drivers.

Due to its broad approach IOP can be enriched in many ways. Our near-future work will concentrate on proving IOP in more industry projects, providing configuration/modeling/design tools, further persistence mappings (MS SQL Server), further protocols (Web Services), and plug-in of specialized integration tools.

References

[Am99] Scott W. Ambler, Mapping Objects to Relational Databases, white paper, AmbySoft Inc., 1999, http://www.AmbySoft.com/mappingObjects.pdf.

[Co01] John Cowan (ed.), XML 1.1, W3C Working Draft, http://www.w3.org/TR/xml11, W3C, 2001

[Cr02] http://www.cronidesoft.com/products/iop.html, 2002

[FHB02] Mohamed E. Fayad, David S. Hamu, Davide Brugali: Enterprise Frameworks Characteristics, Criteria, and Challenges. Communications of the ACM, Vol. 43, No. 10, October 2000.

[ISO99] ISO/IEC 9075-1:1999 Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)

[OMGa] http://www.omg.org/

[OMGb] Workflow Management Facility Specification, OMG, 2000.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch, The Unified Modeling Language Reference Manual, ISBN 0-201-30998, Addison-Wesley, 1999.

[Ro99] Ed Roman, Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition, ISBN 0-471-33229-1, Wiley, 1999.

[Ru01] Craig Russell (ed.), Java Data Objects, JSR000012, Version 1.0, Proposed Final Draft, Sun Microsystems Inc., 2001.

[Sc03] Stefan Schaefer, The Architecture of the Internet Operating Platform, white paper, CronideSoft AG, Germany, 2003.

[WfMC] WfMC TC-1016-P Interface 1: Process Definition Interchange - Process Model, official release, Workflow Management Coalition, 1999.

[W3C] http://www.w3c.org/