Xtractor™
Information Integration Framework
White Paper

Revision 2 - July 2003

Most organizations run ***a number of separated systems****, which* ***contain useful information and data*** *but – for historical, organizational and technological reasons – live* ***isolated and not communicating****. Such systems are "legacy", in the sense that they* ***cannot be easily scrapped away and replaced*** *since they are essential to daily company operations; and* ***yet they have rigid limitations*** *on how, by whom and in what cases the data they contain/manage can be accessed, extracted, and used.*

Xtractor™ Information Integration Framework is a ***software framework for minimizing the effort of building applications that access integrated information from separate data silos*** *within the organization, both for reading and creating/updating information. It does so by providing:*

- *a* ***coherent and effective model for the data and content, based on XML*** *and related technologies (a* ***conceptual model for the data****);*
- *a clear-cut and effective* ***subdivision of the integration process in individual steps****, each with well-defined responsibilities and roles (a* ***conceptual model for information integration****);*
- *a core set of* ***concrete software*** *and tools, each* ***tuned for real-world functional and performance requirements****, which realize the various steps on any hardware/software platform supporting Sun Microsystems' J2EE™ environment (an* ***efficient implementation of the models****);*
- *an* ***extensive support and attention for standards****, which allows for seamless integration of existing technologies and tools.*

The resulting product ***allows to create a single "integrated" applicative view of*** **all** ***the data*** *of an enterprise (or the data that is deemed necessary for a task);* ***to evolve and adapt the applicative view*** *over time (either incorporating more data, adding more functionalities, or both) with minimum effects on the existing IT infrastructure and existing applications; or* ***to change the underlying IT infrastructure*** *without impacting the integrated view and the applications relying on it.*

Several business critical applications have been designed and developed with the Xtractor™ framework *and are currently deployed and running in various enterprises.*

In conclusion, the Xtractor™ framework allows for efficient building applications which access disparate data islands, while preserving the cleanness and coherency essential for their successive maintenance and evolution.

Xtractor™ IIF: rationales and scenarios

Medium and large enterprises typically are organized in several departments or organizational sub-units; each of them providing different services, and having often different needs from software, sometimes processing or using the same type of data in different ways.

In time, such a situation usually leads to a proliferation of different systems serving each sub-unit, using or providing original and/or replicated data, with occasionally overlapping functionalities, specific interfaces and usage patterns, and so on.

As a consequence, **high costs of collaboration and management arise**, overall coordination becomes difficult and expensive, quality of service may suffer or becoming more and more costly. Data cannot be easily exchanged, nor automation possibilities fully exploited; internal bureaucracy spreads – and not (only) for the complexity of the interoperations, but for technological impendence among the various systems.

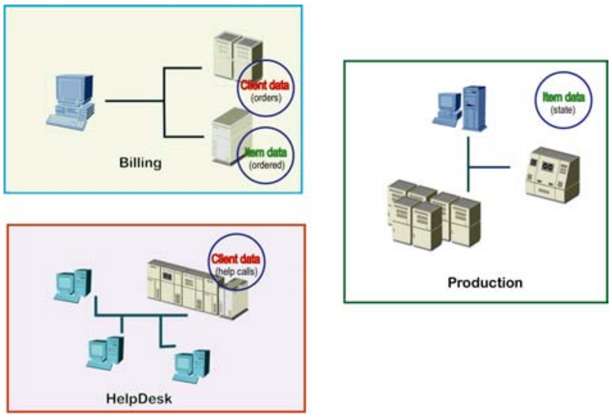


Figure 1 - Isolated data silos in an organization

Being born and grown over different spans of time, such systems are usually running on different platforms and technologies, often mutually incompatible. **Solutions tend to be aimed only to solve specific issues**, little time and resources being usually available for designing and implementing general solutions. **The solutions themselves are not open nor easily extendable**, and being subject themselves to the evolution of the business, quickly end up becoming part of the problem.

Each system (or set of systems) constitutes a separate data silo; and the costs of connecting and integrating such silos are high.

Xtractor™ IIF: design goals

Xtractor™ IIF is built around the fact that systems are changing. It's designed as a **software framework** for building applications[1] that overcome these issues and process and present data independently from where or how it is stored in an enterprise.

By drawing concepts from **data warehousing, relational and object-oriented data modeling** and combining them with the **flexibility of XML** and the **cross-compatibility of the J2EE platform**, Xtractor™ IIF allows for applications which:

- make **use of existing infrastructure** but **do not inherit their technological complexities and incompatibilities**.

  For example, a CRM application can be built around existing customer databases supported by completely disparate systems, without hard-coding the dependencies to the specific systems, or relying on the technology on one specific system either than for communicating with that system;

- allow for **change and evolution**; specifically:
  - updates/replacement of existing system with functionally similar/equivalent ones should not affect the application (or affect it in a minimal and localized way) (**fade-out** adaptability);

    For example, changing the billing system, as long as equivalent information is made available from the new system, will not affect the CRM application, nor require more work than the minimum amount to communicate with the new system;

  - integration of data from other systems should not require expensive maintenance steps (**integration** adaptability).

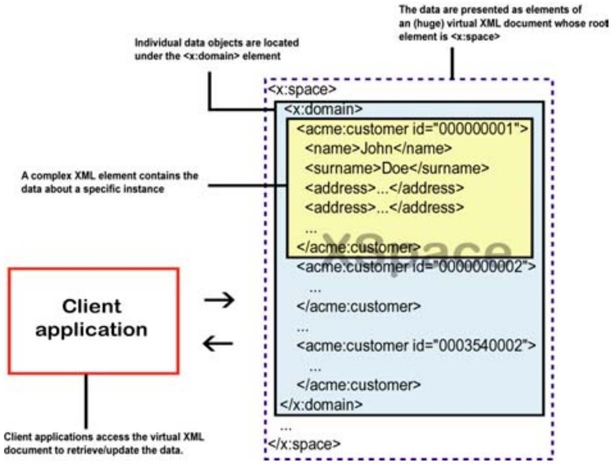
---

[1] Within Sun Microsystems™'s J2EE™ runtime enviroment.

For example, if customer-related data is available from a tracking system (say, the state of the invoicing for some orders), it can be integrated in the CRM application without altering or rewriting the rest of the application;

- satisfy the usual system-level requirements of reliability, no downtime, good diagnostics, efficiency, scalability, etc.

All of this, with a limited amount of resources, time and required skills. In other words, the effect of changes on Xtractor™ framework-based applications is **localized** to specific small components of the application itself.


The data are presented as elements of an (huge) virtual XML document whose root element is <x:space>

Individual data objects are located under the <x:domain> element

A complex XML element contains the data about a specific instance

Client applications access the virtual XML document to retrieve/update the data.

```
<x:space>
 <x:domain>
  <acme:customer id="000000001">
   <name>John</name>
   <surname>Doe</surname>
   <address>...</address>
   <address>...</address>
   ...
  </acme:customer>
  <acme:customer id="0000000002">
   ...
  </acme:customer>
  ...
  <acme:customer id="0003540002">
   ...
  </acme:customer>
 </x:domain>
 ...
</x:space>
```

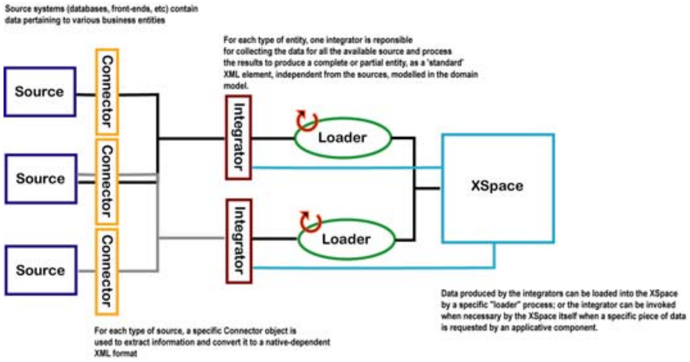
Figure 2 - the XSpace document

Xtractor™ IIF: realizing the goal

The Xtractor™ framework realizes these goals exploiting two main ideas:

The first idea consists in modeling all the data available to an organization as **a huge (virtual) XML document**, collectively referred to as the **XSpace** document. This document is structured in such a way that each piece can be pinpointed, collected and combined for presentation or further processing.

One part of the document (the **domain model**, located within the complex element **<x:domain-model>**) describes the data from a business perspective.


Source systems (databases, front-ends, etc) contain data pertaining to various business entities

For each type of entity, one integrator is reponsible for collecting the data for all the available source and process the results to produce a complete or partial entity, as a 'standard' XML element, independent from the sources, modelled in the domain model.

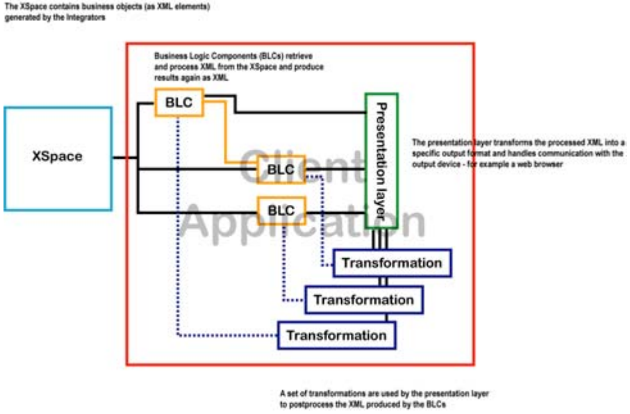
Data produced by the integrators can be loaded into the XSpace by a specific "loader" process; or the integrator can be invoked when necessary by the XSpace itself when a specific piece of data is requested by an applicative component.

For each type of source, a specific Connector object is used to extract information and convert it to a native-dependent XML format

Figure 3 - Populating the XSpace document

Individual items of data (which collectively make up the **domain** of the application) are made available as complex XML elements under a specific **<x:domain>** element.

Database rows, applicative documents (for example, a Microsoft Word document or an email) and any relevant metadata can be stored in the domain, thus becoming part of the XSpace document as XML, and immediately accessible by applications built on top of it.


The XSpace contains business objects (as XML elements) generated by the Integrators

Business Logic Components (BLCs) retrieve and process XML from the XSpace and produce results again as XML

The presentation layer transforms the processed XML into a specific output format and handles communication with the output device - for example a web browser

A set of transformations are used by the presentation layer to postprocess the XML produced by the BLCs

Figure 4 - application logic over the XSpace document

It's worth mentioning that albeit **the XSpace can be programmatically accessed** as an XML document (for example, locating a specific element identified by an XPath[2] expression; traversing elements and sub elements; replacing nodes; processing it or part of it with an XSLT[3] transformation; and so on), **it is not actually implemented as an XML file[4]** – since it can contain literally millions of nodes.

---

[2] XML Path language, a concise and standard syntax to identify one or more nodes in an XML document.

[3] XML Stylesheet Transformation Language – a standard language, itself syntactically written as XML, to transform XML in any target format.

[4] Which is the physical support most XML documents usually reside in.

Rather, different parts of the XML document are supported by a range of physical implementations – from Xtractor™'s own XDB, an high-performance transactional XML storage system built over a relational DBMS (see ***Xtractor XML Storage (XDB) White Paper***), to in-core caches and so on.

Xtractor™ IIF: the information integration process

The second idea is to define clearly **the process that populates such XML document** and concretely makes the data available to applications.

Data is first **extracted** from a source (database, filesystem, network server, etc) and pre-processed in an intermediate source-dependent XML; then it's **integrated** towards a common source-independent XML and **made accessible as part of the XSpace document**; finally, the XML in the XSpace is **processed by some business logic**, which thus becomes naturally independent from the original location and format of information.

XSpace as an XML document – why

- Consistent and unified access.
- Easy to understand and learn.
- Few skills required for using Xtractor IIF efficiently (XML and XPath, and Java and/or XSLT if necessary).
- Easy to distribute load across several Xtractor IIF server instances, without altering applicative code.
- Configuration and other metadata are simply part of the XSpace document itself.
- Queries are not limited by the data available in one source, or the capabilities of one source
- Application load is better distributable over time between the original source and the XSpace – peak activity on the application does not necessarily impact on the XSpace.

Connectors – why

- Data is seen as XML independently from the source format
- Easy adaptability to update/changes in the sources (for example, new versions of products like MSWord, etc)
- Decoupling connection from integration allow to change/substitute the sources without breaking the integration logic.

In particular, by clearly separating the extraction of data from its integration and again these two from the actual business logic, the Xtractor™ framework maximizes the chances to develop applications which will adapt efficiently to changes in their domain models and/or in the physical data sources.

Also, the integrated XML can be made available as part of the XSpace document by either a **periodic loading process** or integrated directly **on an on-request basis**. From a development point of view, XSLT is often a perfect candidate to implement integrators - albeit both Java and a range of other programming and scripting languages can be used.

The application logic is located in a number of **Business Logic Components (BLCs)**, which have access to the XSpace or to each other – and implement a specific functionality – producing (again) XML as results: again XSLT – or a mix of XSLT and Java – are often the best candidates for expressing the business logic in concrete code .

Integrators – why

- Integrators allow for a complete, updateable and business-view of data reusable through different applications
- Allows for complex correlation among related pieces of information residing in different sources.
- Information can be easily validated and "cleaned up", or filtered out at integration time.

Presentation is decoupled from the business logic by being delegated to a specific layer, which transforms the XML produced by the BLCs into formats suitable for a specific output device (typically, but far from only, a web browser).

Xtractor™ IIF: the architecture

BLCs/Presentation – why

- BLC operates on integrated entities, independently from the original format/sources
- BLC allows for encapsulating business logic in reusable components
- A separate presentation layer abstracts the results of the business logic from the actual output format

Each of the steps discussed above corresponds to a logical layer in Xtractor™ architecture. For each layer, concrete implementations, object libraries, and/or components are provided as part of the Xtractor™ software. Some of them are fully-functional configurable components[5]; other are libraries or partial implementations, from which application-specific development can start quickly.

The **XSpace** ***document*** is the central point of the architecture, consisting of a (virtual) XML document containing the integrated information to be processed by applications (very much like a data warehouse or a traditional database). Within the XSpace document, the functionality and contents of each layer are accessible as sub elements of the root element **<x:space>**.

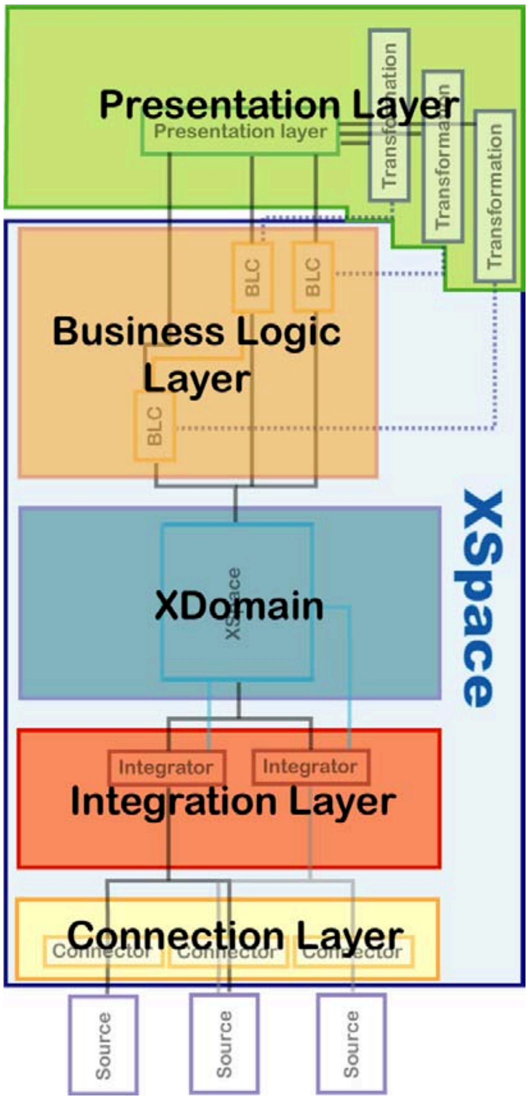


Figure 5 - the layered architecture of Xtractor IIF

The **XSpace** ***component*** is the corresponding software object (realized as a J2EE Enterprise Java Bean) which encapsulates access to the XSpace document and is responsible for making all the other components/layers appear as they are part of it.

The XSpace component implements an interface that exposes services to

- **retrieve an XML fragment** from the XSpace by using an XPath expression;
- **insert/update an XML fragment in the XSpace**, by providing the fragment plus an XPath expression to indicate the insertion point.

This interface is made available both to Java/C++ and other scripting or declarative languages (XSLT, JavaScript, etc).

XSpace/Domain

Xtractor™'s **domain** is the section of XSpace document where the actual data can be found or inserted/updated. It consists of all the XML nodes under the location path **/x:space/x:domain**.

This section is usually supported by a high-performance XML storage system, populated with data provided by the Integration layer, and accessed by business logic for further processing. The domain can be configured to automatically update itself, either on a regular schedule or by setting an update period on specific fragments of information.

After an initial setup phase, most of the querying and updating is done over the XSpace domain.

Configuration and administration of the composition of the organization-specific instance of a specific instance of the XSpace document (and therefore of the services made available by the XSpace component) can be executed from an user friendly web interface.

The XSpace component is the core of Xtractor™ IIF: a fully functional EJB-based implementation backed by our high-performance XML storage system, provided as a standard component, which encapsulates access to the XSpace document. Configuration occurs via a specific web application which allows to examine and edit the contents of the XSpace document. Bridges to languages other than Java™ (XSLT, JavaScript, etc) are provided as well.

---

[5] Typically, usage-ready connectors are provided for a number of different source types, ranging from relational databases, files, MSOffice formats, internet protocols and so on – see "Connector layer".

Connector layer

The **Connector** layer is responsible for translating XPath expressions into queries and commands to and from existing sources of information and converting source-specific formats to and from (source-dependent) XML fragments.

As an example, a database connector allows to "see" a database table as a set of XML elements (whose element names are dictated, for example, by the column names in the table) and to retrieve a certain number of such elements by indicating an appropriate XPath expression. For example, an SQL row like the following (in a table **`PERSON`** on a database **`CUSTOMERS`**):

ID	NAME	ADDRESS
000001	John Doe	Cherrylane 25

can be connected by a database connector[6] and retrieved by an XPath expression like **`/x:connector/customers.person[id='000001']`** will produce the following XML[7]:

```
<customers-person>
 <id>000001</id>
 <name>John Doe</name>
 <address>Cherrylane 25</address>
</customers-person>
```

The connector layer effectively hides all the details of connecting to a specific source (host names, SQL's, program statements, file-placement, etc), allowing the successive layers to concentrate only on processing the information.

Several connectors are provided as a standard part of the Xtractor™ library:

- relational databases (via JDBC™)
- MSMQ
- MQ Series
- LDAP
- Active Directory
- Http/Ftp and other TCP protocols
- HTML
- generic file system
- XML files
- MSOffice files
- MS Project
- Agresso
- Autocad
- Image formats
- Nortel Meridian MAX
- Nortel Meridian v2.5
- other native formats

Plus, a documented connector library to develop (or adapt existing) system-specific connectors in Java/C++ is provided.

Connectors are easily configured through the Xtractor Console

Integration layer

The integration layer's responsibility is to coordinate the work of the connector layer by translating source-dependent XML into source-*in*dependent XML suitable for populating the XSpace, and vice versa.

An integrator is typically dedicated to a specific business object. It knows which sources (i.e., connectors) contain data about that business object, **how to collect them** by using connectors, and **how to correlate them** so that instances of the business objects can be aggregated (as XML) or disaggregated as necessary.

---

[6] Properly configured.

[7] With some additional XML details here not shown for clarity.

For example, one database may produce **<db_name>** as XML tag for the element containing the textual information about the name of a person; another database, containing additional data about that person, might be producing the tag **<personal-identification>**.

If the domain model states that the 'name' attribute of the business object "person" is identified by a **<name>** tag in the corresponding XML, it is responsibility of the integrator for 'persons' to correctly match and verify the contents **<personal-identification>** and **<db_name>** elements so to produce an unique **<name>** element in the final XML (or to warrant against inconsistencies).

> Integrators are inherently application-specific: Xtractor™ provides a tool (Xtractor Console – Integration Tool) to support the process of building integrators.

Business Logic layer

This layer generally contains any business logic which, together with the integration and presentation layers, composes an application.

It's responsibility of this layer to locate and elaborate information in the XSpace document to implement the functional requirements of a given application, producing application-specific XML which will be further processed by the presentation layer.

For example, if a report on "relationship between customer calls and purchased products" is required, a specific BLC can be built to process the data in the XSpace and produce the results; or a generic "report engine" BLC can be built to serve different report requests.

Several architectural examples are provided in the **Xtractor™ User Guide**.

> Again, BLCs are highly application-specific.
>
> Xtractor IIF provides a framework and an administrative tool for developing BLCs.
>
> For XSLT based BLCs, a documented library of extension functions (see **Xtractor™ XSLT Library User Guide**) for common tasks related to accessing the XSpace component and processing data from the XSpace document is provided.
>
> Xtractor™ IIF also provides a pre-built framework of classes for building or connecting existing business logic in Java/C++ and every language compatible with the Java platform.
>
> Specific ready-made implementations are also included, that allow expressing the entire business logic with hosted scripting languages (XSLT, JavaScript etc).
>
> Finally, Xtractor™ IIF comes with a set of tools and use-ready sample applications for accessing the X-Space XML by XPath from the web and performs routine tasks.

Presentation layer

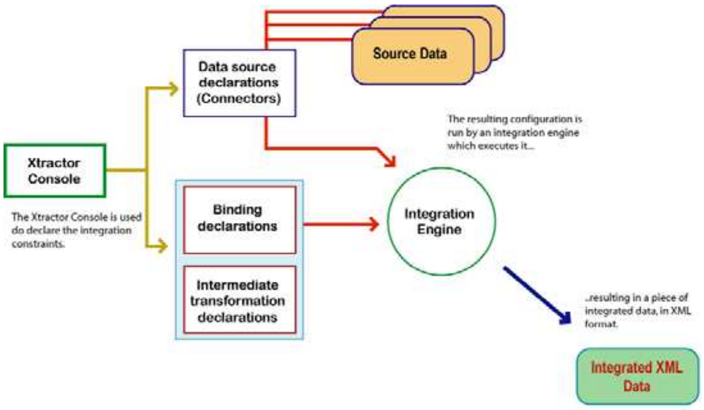
This layer provides presentation mechanisms by which the XML produced by the BLC layer can be properly converted in other formats (say, HTML for web presentation).

> A presentation layer aimed to web applications ("**Visualizer**") is a standard component in the Xtractor™ framework: it consists of a configurable servlet, running in any Servlet-enabled HTTP server, whose responsibility is to invoke one or more BLC and process the resulting XML with some result processing logic, typically XSL transformations, producing HTML to be send back to a client browser.
>
> Xtractor provides a Visualizer tool that enables easy web presentation of information from the Xtractor IIF.

Declarative integration

A key element of the Xtractor™ approach is that, in a large number of cases, integration can be performed by simply **declaring** the properties of the data bits to be integrated (such as which data sources to use, which data pieces are logically **binding** data found in these source, which intermediate transformations are necessary, and so on).

Therefore, whoever is building the integration logic needs only to:



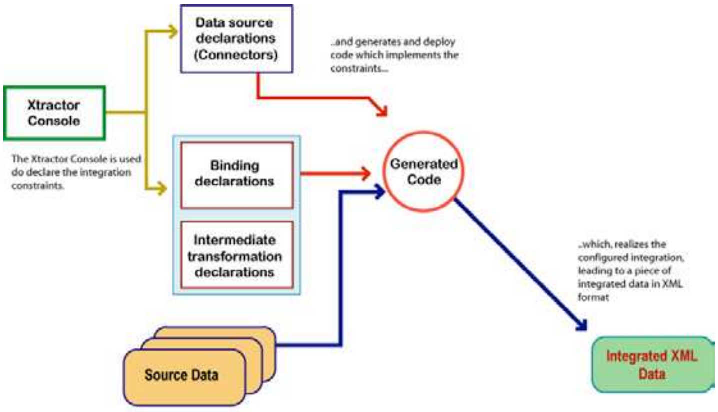
Declarative Integration - Logical view

- **configure** the connector layer (thus declaring which sources are available and to use) ;
- **declare** which XML elements produced by the connectors (which are equivalent to SQL columns, file lines, LDAP record entries, etc, depending on the specific connector) are tying together the data (for example, a certain SQL <ID> column in a table found in a database source corresponds to an USER-ID field in an LDAP record found in an LDAP source);
- **declare** any intermediate logical step (for example, the <ID> column might not map directly to USER-ID, but require a lookup in another database source containing a map ID/USER-ID);

Since all the data in the various sources are accessible as XML **independently from their original format**, as provided by the connectors, such declarations amount to define correspondences between XML element/attributes in different pieces of XML.

From a *logical* point of view, such configuration is then feed at runtime to an *integration engine*, which actually performs the integration job –by starting from one source and following the various bindings until all the data bound together has been found – therefore producing the integrated data.

However, for performance reasons the actual *physical* code executing at runtime is pre-generated at some stage (typically using the **Xtractor™ Console**, see below) as XSLT or other type of code.



Declarative Integration - Physical view

Therefore, at runtime, there is no impact on performance due to the interpretation of the integration declarations.

Xtractor™ Console

Xtractor™ has implemented a **development console** allowing a user with no programming background[8] to:

- *configure* the connector layer (which sources are available, of which type, etc);
- *configure* the integration constraints (binding between source XML provided by different connectors, type of integration logic required, aggregation, writeback to sources etc) – and have the specific code (XSLT) generated and installed by the console;
- *configure* the visualizers (mapping from data to GUI)
- *configure* **value adding services** such as security filter, cache, journal, timeout, writeback etc.
- *setup* a **loading process** which actually runs the generated code;

---

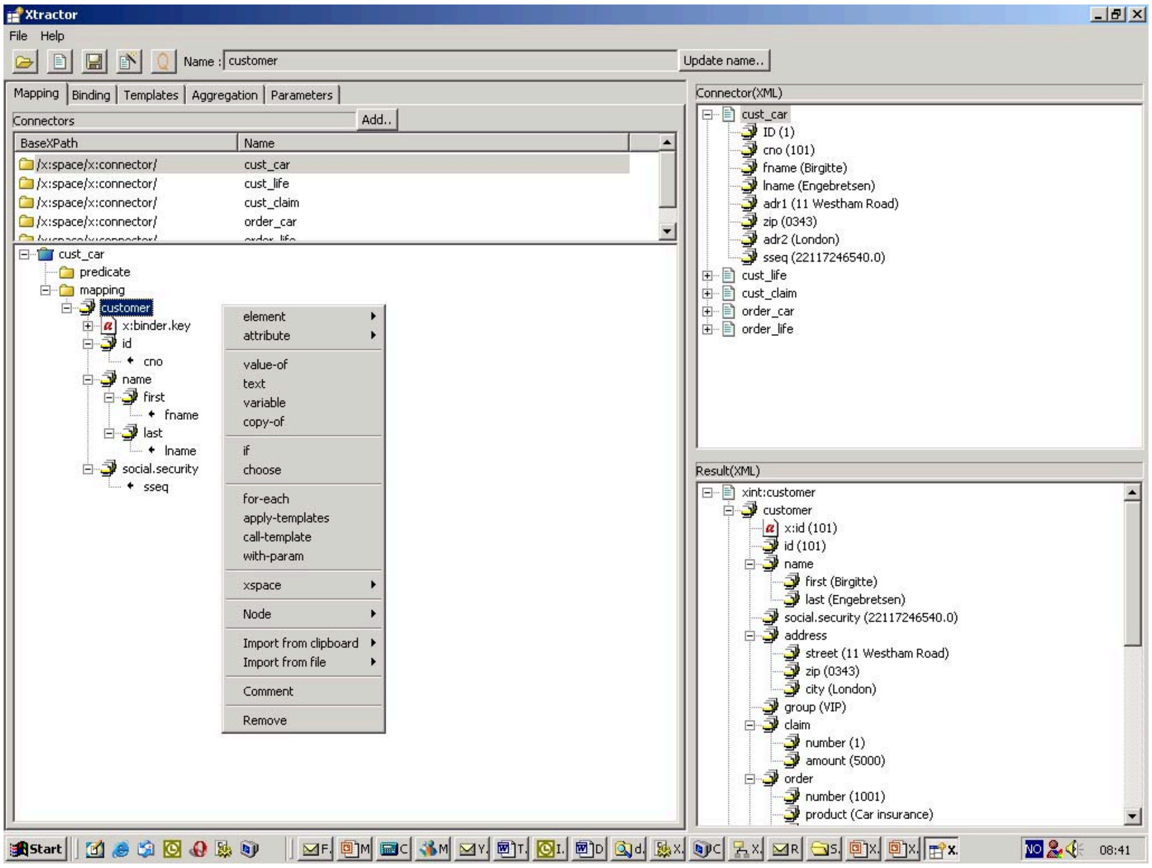
[8] But knowledge of the business domain and knowledge of the Xtractor™ framework layers.

- *documentation* of configuration and logic
- otherwise control and configure the various layers composing the XSpace.

Overview: an Xtractor™ application

In summary, an Xtractor™ IIF integration foundation is built by:

- **collecting the functional, use case and system requirements** for the application as usual;
- **modelling the business objects**, creating an initial definition as (descriptions of) complex XML elements[9];
- **attaching the necessary connectors** to communicate with the systems to integrate;
- **configuring integrators** which collect data (using the connectors above) from the sources and transform them into the XML; such integrators can be developed visually either by employing the Xtractor™ console or; as pieces of programmed code.
- **configuring the XSpace and the loading/update** processes for each integrated object, via the available tools;



The Xtractor™ Console simplifies and streamlines these steps in most cases.

An application is built over such foundation by:

- **developing business logic components** to serve the processing requirements of the application;
- **configuring presentation transformations** to visualize/communicate the results of the business logic execution.

The application lifecycle is then managed by

---

[9] This can be done by using DTDs, XSchemas or just creating examples of XML elements containing the proper data.

- **updating the business object models as necessary**, to include more information whenever new data sources become available. Since the business objects are expressed as XML, this doesn't usually require any rewriting of the existing logic;
- **adding and configuring the required additional connectors**, if any;
- **updating the integrators** for the business object being expanded, to integrate the new data;
- **adding/update the business logic** to make use of the new data;
- **adding/updating** the presentation layer to include the new data;

Xtractor™ IIF: a complete information integration framework

It's worth to remark that Xtractor™ IIF is a ***complete*** framework: typical **applications can be constructed entirely with the standard software**. However, a component-oriented architecture and clear responsibilities/interfaces between each layer ensure that **external components can be easily plugged in if deemed necessary**.

Time-to-market efficiency is optimal, as is adaptability to the skills available in the specific organization, due to the wide range of programming and scripting languages supported, and the adherence to well-known and widespread standards.

In conclusion, as a result of a purpose-oriented architecture and a careful implementation, Xtractor™ IIF constitutes an excellent framework for developing cross-system applications, dramatically lowering costs and overheads due to technological impendence between systems.