

1 Overview

This document describes MONDO, a generalized architecture for encoding, modeling, and processing information. MONDO is the result of evolving and integrating the concepts from descriptive markup with the concepts from object-oriented information modeling. This produces a very flexible and powerful system for working with both structured documents and human-readable information models, and removes the boundaries separating them. The techniques and tools from multiple industries can be focused on common problems.

MONDO is very general and can be used with many different types of projects. The following are some of its current or possible applications:

1. Structured Document Management
2. Object Serialization and Interchange
3. Hypertext
4. Literate Programming
5. WWW Sites
6. Program Properties and Localization
7. Data creation and backup for Information Systems

Some of these applications may sound repetitive. That is because they are, but in the past most of the applications have used completely different tools. MONDO uses a single architecture and provides core functionality to handle the common needs of these applications.

On a more fanciful level, some of MONDO's concepts may have much broader impacts when combined with other technologies. These could include language-independent object agents moving around the internet, ultra-large knowledge bases instead of the textual WWW, the wide-spread reemergence of LISP with a different syntax, or a dramatic simplification of CORBA like technologies. These are the Blue-Sky & Blue-Plane thoughts regarding MONDO.

But for now, MONDO will try to do a few important things well, integrate with as many good technologies as possible, and see where this all leads.

1.1 Organization

This document is organized to first introduce you to the MONDO concepts and design and then go into more detail about each of its components. This introduction and detailing occur in the first nine chapters. Chapter 10 shows how the MONDO architecture can be applied to specific tasks that are from totally different fields. That MONDO can support these different tasks well shows that it is a successful architecture for existing applications.

The subsequent chapter contains blue-sky thoughts and proposals for how MONDO could be useful in some non-obvious and some unexpected ways. The document finishes with appendices that contain end notes, a glossary, suggested reading, and the OML syntax.

1.2 Prerequisites

A lot of patience. The MONDO project integrates several dense and mysterious fields that each has its own terminology. It will be rare to be familiar with all the terms even if they are well defined within each of the fields. Add to this MONDO or ChiMu specific terms can make the document hard to follow. I can only hope we have done a good job at using the most accepted terminology and explaining new terms. The suggested reading chapter can direct you to fuller explanations of the concepts that are referred to and integrated within MONDO.

The MONDO WWW site at:

<http://www.chimu.com/projects/mondo/>
contains FAQ's about MONDO and status information on the project.

This document is not tailored to a specific background nor is it gentle. We are planning to do SGML, OO, and LISP oriented introductions to MONDO but none of these will be ready anytime soon. A Gentle Introduction to MONDO would be nice too (even if it sounds like an oxymoron) but is not on the horizon. We are hoping that the MONDO design is good enough and simple enough that it is gentle even if the documentation is not.

1.3 Supplemental Reading

The suggested reading chapter provides a selection of publications that are very relevant to MONDO. This list is incomplete but still very long. If you could only read one book from each section I would have to choose the following for their diversity, relative accessibility, and depth in areas related to MONDO:

| | |
|--|--------------------------|
| <i>Literary Machines</i> | Nelson 81 |
| <i>Object-Oriented Modeling and Design.</i> | Rumbaugh+BPPEL 91 |
| <i>Developing SGML DTDs: From Text to Model to Markup.</i> | Maler+A 96 |
| <i>Structure and Interpretation of Computer Programs.</i> | Abelson+S 96 |

1.4 Status and Feedback

This is a partially finished initial public draft of this document. Please be kind. There are missing chapters and missing sections within chapters. Most of these should be finished by the beginning of December. Information on the status of the MONDO project can be found at the MONDO WWW site.

Any and all feedback would be highly appreciated. Please send them to:

mark.fussell@chimu.com

or

mondo@chimu.com

1.5 History

- v0.3.1 971127 Added sections 4.2 through 4.5 (Building and Recipes), fixed the conclusion of chapter 5 and added a comparison table. Cleaned up chapter 10.
- v0.3 971125 First public release to XML-Dev and c.t.sgml

1.6 Acknowledgements

MONDO is mostly other people's work. MONDO is the result of unifying concepts from information modeling, structured document processing, object-oriented and functional programming, and many other fields. All the hard foundation work for MONDO was done by the people in those fields. Some of these people are listed in the references but there are many, many more. I am grateful to all of them.

Special thanks must go to the progenitors of the concepts which are used within MONDO: Alan Kay, Charles Goldfarb, E.F. Codd, John McCarthy, Kristen Nygard, C.J. Date, Guy Steele, and Donald Knuth. I feel heavily indebted to their great thoughts, their writings, and the resulting work that they have caused.

I will also define terms and concepts as part of MONDO but the reader should be aware that these terms are usually borrowed from some other field. It is impossible to explicitly acknowledge (or even identify) each source for each term during the body of this document, but I recognize that the work is not original and the reader may want to look at the references to search for the original conception.

2 Introduction

MONDO provides a generalized architecture for encoding, modeling, and processing information. It does this by evolving and integrating the concepts from descriptive markup with the concepts from object-oriented information modeling. The result is represented by two core concepts: DomainObjects and Recipes. These core concepts are integral to everything MONDO does and they allow MONDO to take advantage of many other technologies and disciplines. The power of these simple concepts, a simple but flexible architecture, and the leveraging of multiple other industries allow MONDO to be very capable. This chapter will introduce you to MONDO's core concepts, its general architecture, and the beginnings of its capabilities.

2.1 Core Concepts: DomainObjects and Recipes

MONDO has two core concepts: DomainObjects and Recipes. These provide the foundation for reasoning about how MONDO works and what MONDO's components are doing. "DomainObject" is a well-defined term within Object-Oriented Information Modeling¹ and MONDO uses the same definition. "Recipe" has a very specific definition to MONDO but the concept has been in computer programming since before LISP. MONDO's recipes also resemble the popular concept associated with cooking, so they benefit from both a formal programming origin and a day-to-day naturalness.

DomainObjects

DomainObjects embody, inside a computer, knowledge about a particular domain (subset) of the world. They allow a computer to inspect, imply, modify, and "reason" about that information in either very simple ways (the facts) or more complex ways (the rules and implications). The capabilities of DomainObjects depend on the sophistication and focus of the model (the **DomainModel**) of the information. The DomainModel could be just a simple data structure or could be a very sophisticated Knowledge model. The DomainModel concept is the same in either case: it is the information, operations, and rules used to represent a subset of knowledge about the world within the computer.

Recipes

Recipes are instructions for building knowledge. Specifically, recipes are the instructions for building DomainObjects, which together will represent knowledge within a computer. A Recipe does not describe the resulting DomainObject. The resulting DomainObject is determined by the collaboration among the DomainModel, the ObjectBuilder*, and the recipe itself. These together will create a DomainObject that instantiates the knowledge in a recipe in a fashion appropriate for the DomainModel, and that resulting DomainObject could know much more about the "world" than the single recipe would have indicated.

This is similar to the cooking analogy: The ultimate dish (DomainObject) is determined by the collaboration among the Restaurant (DomainModel), the chef (ObjectBuilder), and the recipe itself. If you give a restaurant a collection of recipes to prepare you will get dishes that are far more intricate than the recipes themselves[†].

MONDO Recipes have several distinguishing characteristics that make them valuable:

8. **Recipes encode knowledge.** The purpose of a recipe is to encode how to build knowledge within a computer. That knowledge can then be used for many different purposes.
9. **Recipes are simple and self-describing.** Recipes are simply trees of recipes and names. They are almost as simple as lists and they are more self-describing. The textual encoding of a recipe requires fewer than 5 special characters (OML uses ' <>=' as its core). The rest of a recipe is described in terms of recipes themselves.

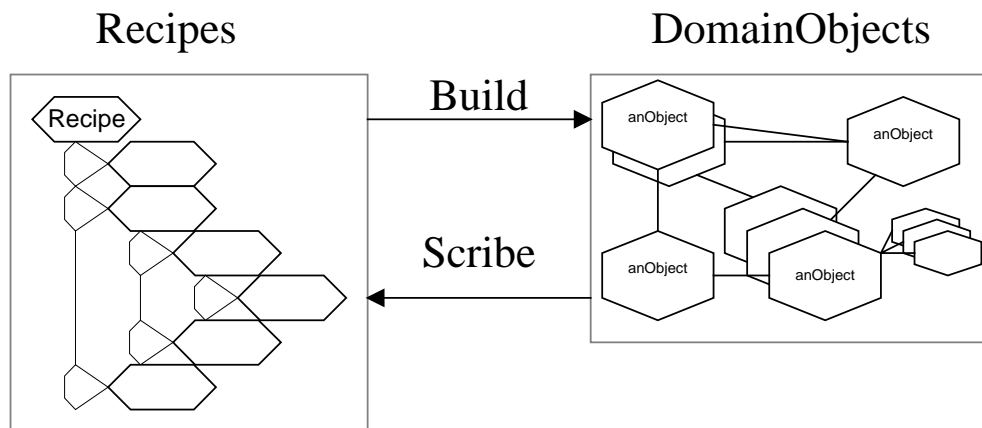
* ObjectBuilders will be formally introduced shortly, but they are the component that builds DomainObjects from Recipes. Similar to a chef.

† And if you give this author a recipe you may get much less "fidelity" than you expected but it will probably keep him alive.

10. **Recipes are accessible.** It is easy for any person to read and write recipes without needing sophisticated tools. Recipes can be entered with simple text editors and read off of paper.
11. **Recipes are application independent.** Recipes can be used by many different applications for different purposes. They are designed to just encode information, not the processing to perform on it.
12. **Recipes are understandable.** We believe it is easy for a person to understand simple recipes and comparatively easy to understand sophisticated ones. This is because recipes have a clear objective (building knowledge) and simple analogy (cooking food).
13. **Recipes are not a new programming language.** The encoding of recipes and the potential capabilities of a MONDO ObjectBuilder are general enough to be a Turing complete programming language (recipes are like a slightly more structured LISP syntax). This is a side effect of the unification and simplification of concepts that were part of MONDO's goals. Although this is a useful capability, recipes focus on encoding knowledge not processing.
14. **Recipes are programming language independent.** Recipes encode the information on how to build knowledge with object-oriented information structures. Any language can be used to implement these concepts and read recipes. It is in the spirit of MONDO to not encode language specific details within the core recipes. These details are either left up to the application or associated with the core knowledge through metadata recipes.

Transforming between DomainObjects and Recipes

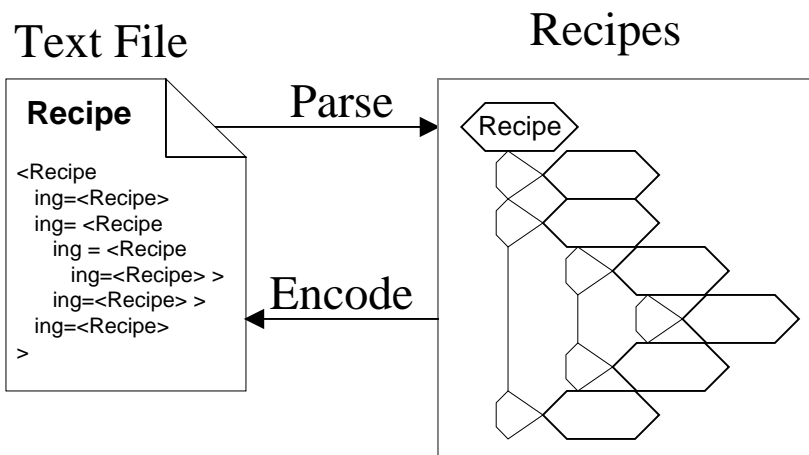
The construction of DomainObjects from Recipes is called **Building**. The process of inspecting DomainObjects and turning them into Recipes is called **Scribing**. The details of these processes will be discussed in the next chapters.



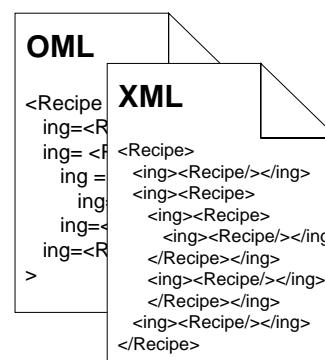
Encoding Recipes

The Recipe represented above is an abstraction within the computer. We also need to expose them to people: we need to provide the ability to read and write recipes. Because recipes are a simple abstraction there are many ways to put them into a computer and it is simple to make computer programs work with them. To make recipes highly accessible to people and widely supported by computers, MONDO can encode and decode recipes from simple text files.

MONDO encodes recipes into text files in a manner like SGML. Text files have very simple “markup” that indicates the structure of the information in that file. SGML has shown that using marked-up text works very well because it is highly-accessible to any user and it can encode arbitrarily complex information quite easily. The correspondence with SGML continues because MONDO uses SGML/XML as one of its primary encoding formats. MONDO also uses a slightly different markup language called OML (Object Markup Language) which is very similar to SGML but has a more direct translation to recipes. These two formats are expected to be the primary means of encoding recipes in human-readable files.



The process of creating recipes from a text file is called **Parsing** (or **Decoding**) and the process of creating text files from recipes is called **Encoding**. The transformation between Recipes and TextEncoded-Recipes is meant to be very direct. You should feel you are directly editing the Recipe when you modify a text file and that you are directly seeing the recipe when you look at the text file. This is the ideal of encoding. For OML, this directness of translation is present. For SGML/XML, there may be very minor transformations to support preexisting documents. In all cases, the Recipe is the true information and you can easily translate between one recipe format to another.



Creating Recipes for Knowledge

A Recipe needs to capture the essence of knowledge and how to build it. How do you do that? Unfortunately, there is no right answer. It dramatically depends on how general you want the information to be, how long you expect to the information to last, and what other knowledge is already available (e.g. as Recipes or in other forms). There are many domain-specific and general resources for understanding what information to model and record. See the references section of this document for an initial set of suggested reading and sources to help with information modeling, analysis and design patterns, SGML DTDs, and example documents and systems.

The important part is the process: try to focus on how to represent knowledge in as general and useful way as possible. Try also to reuse as much of others people's work as possible. These are core principles to the construction of MONDO itself. MONDO can not determine what is worthwhile about a domain but Recipes and DomainObjects provide the spirit of the process and MONDO provides mechanisms to very flexibly encode and work with that knowledge.

Summary of Recipes and DomainObjects

DomainObjects embody the information, rules, and operations that are needed to represent knowledge within a computer and to work with that knowledge. Depending on an application's desired functionality, DomainModels can vary from simple data structures (e.g. SGML's Grove structure) to sophisticated knowledge bases. Each of these models can support many applications that will interact with and process the domain knowledge.

Recipes are even more abstract and general than DomainModels: They simply encode knowledge. Recipes describe how to build knowledge within a computer by constructing and associating DomainObjects. A

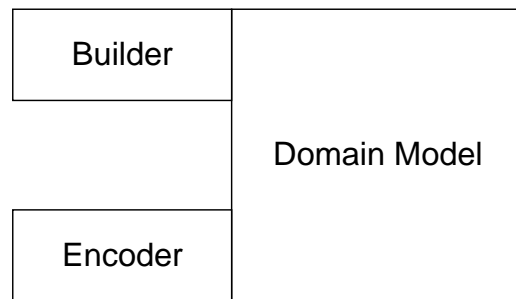
Recipe can be used with multiple different applications having different DomainModels on different platforms. Users can directly interact with recipes because recipes directly translate to text files as OML or SGML/XML.

Recipe and DomainObjects are the crucial concepts to MONDO. All of the MONDO architecture is there to support working with them. Object-oriented DomainModels provide a very general information modeling approach that can handle documents, business models, object structures, and many other forms of information much more expressively than other approaches. Recipes provide a way to encode information that is independent of both the application and even the level of representation detail of the DomainModel. Recipes do not describe information, they describe how to build it.

The rest of this document will reveal more of the details about DomainObjects, Recipes, and how the MONDO architecture supports working with them.

2.2 Subsystem Architecture

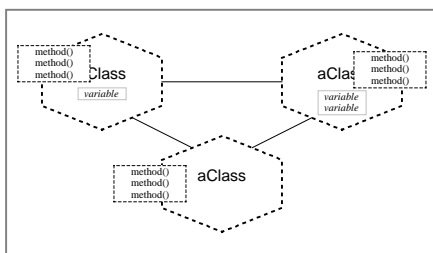
MONDO has three major subsystems: the **ObjectBase**, the **ObjectBuilder**, and the **ObjectEncoder**. The ObjectBase is the core component: everything MONDO does is focused on building, processing, supporting, or encoding objects. The ObjectBuilder puts information into the ObjectBase and the ObjectEncoder externalizes information from the ObjectBase. MONDO has interfaces, structure and functionality supporting all three of these subsystems. MONDO also frequently does very little, and instead works with or relies on other people's functionality.



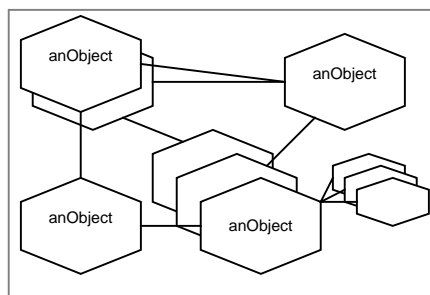
ObjectBase

An **ObjectBase** captures the knowledge, operations, and rules required to usefully represent a particular part of the world in a computer. These responsibilities are divided between the DomainModel*, which captures the static properties (e.g. associations and operations) of the ObjectBase, and the DomainObjects, which capture the current dynamic state of the ObjectBase.

DomainModel



DomainObjects

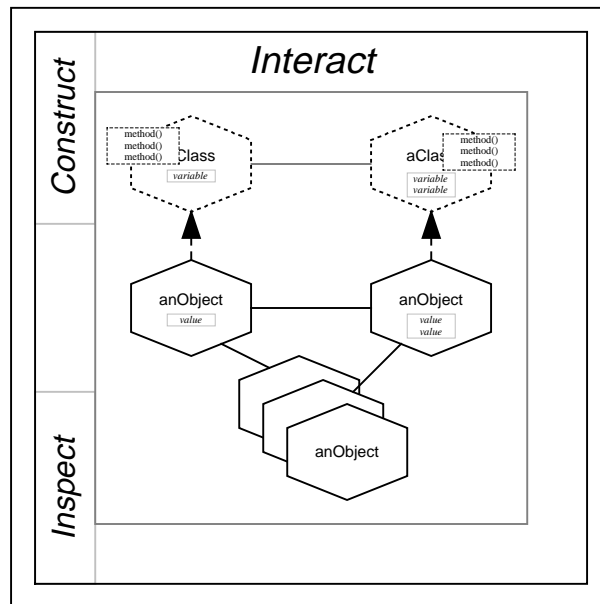


Together these form the full ObjectBase. In later chapters we will discuss the functionality that MONDO needs or would like from the DomainObjects to perform its services. In this overview we will say that MONDO needs a way to **construct** objects to build the ObjectBase and to **inspect** objects if you want

* For MONDO, the DomainModel need only be conceptual. MONDO does care whether the model is implemented by Classes or whether it is produced dynamically based on the Objects themselves. MONDO only requires the ObjectBase to be able to build Objects.

MONDO to be able to create recipes from them. Finally, your application must be able to **interact** with the objects after construction to query their knowledge or change their state.

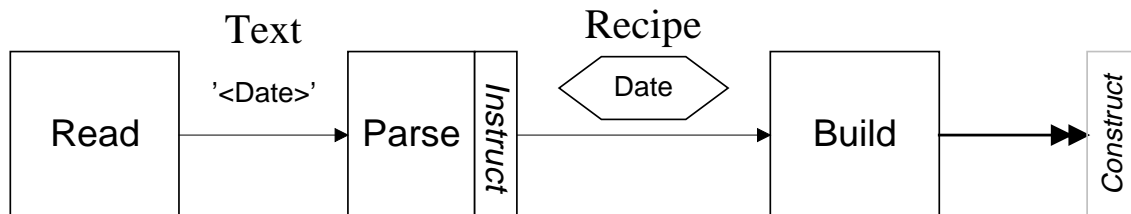
Objectbase



ObjectBuilder

The responsibility of the **ObjectBuilder** is to build all or part of the Objectbase from an external source. Generally this source will be a human-readable text file, but there are several stages to ObjectBuilding which can each have different approaches (e.g. we could read from a binary file instead). Assuming we have a textual file-based approach, ObjectBuilding would go through three stages:

1. **Read** from the text file and produce a stream of text
2. **Parse** the text and turn it into a **recipe** (what objects to build and what ingredients to use)
3. **Build** the recipe and **construct** objects within the ObjectBase

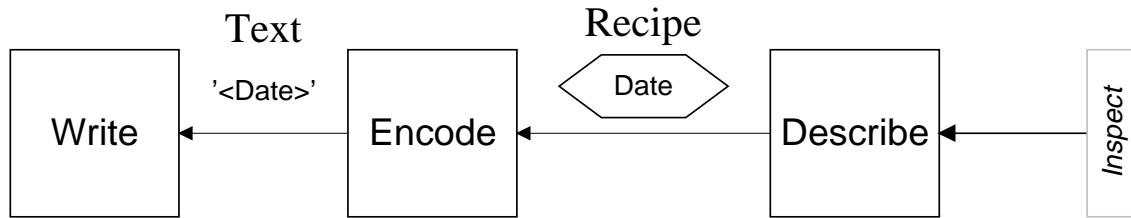


The details and options of this process will be discussed in the upcoming chapter.

ObjectEncoder

The responsibility of the ObjectEncoder is to save sufficient information about an Objectbase to an external repository so that a similar Objectbase can be recreated later through an ObjectBuilder. Again, the external repository will generally be a human-readable text file and the process would go through three stages:

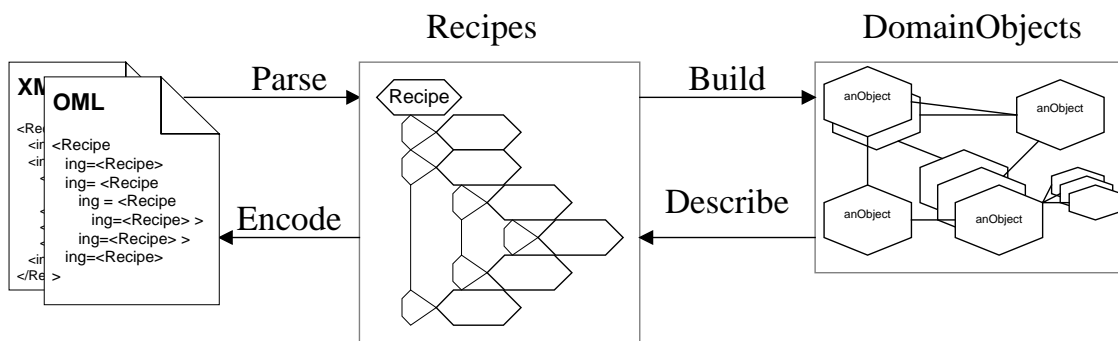
4. **Inspect** the domain model and **describe** it into a recipe
5. **Encode** the recipe into a particular textual format
6. **Write** the text into the text file



The details and options of this process will also be discussed in the upcoming chapter.

Recipes

Although not a component within the architecture, recipes are integral to the building and encoding process. They also provide the ability to hook the building and encoding stages with each other in alternative patterns. For example, an ObjectBase can be migrated to a new ObjectBase by Describing and then Rebuilding. Another option is to migrate between two Recipe Encodings by Parsing and then Re-encoding.



An important aspect to Recipes is that they are usually virtual: instead of being explicitly built and then giving to the next component, they are described implicitly by the communication between components (e.g. the Scribe and the Encoder).

Modeling and Processing

MONDO is designed to build and encode ObjectBases. Along with these primary goals, MONDO will describe (and MONDO implementations provide) features and interfaces to support modeling capabilities and processing available on the ObjectBase. These will include object visiting, hyper-retrieval, queries, validation, introspection, and metamodeling. Other capabilities may be added under the MONDO architecture or certain of these features dropped.

The important principle is that *functionality is always added to the ObjectBase*, so other applications can take advantage of them or other applications can provide them if MONDO does not. The responsibilities of the Building and Encoding components are well defined and restricted. Application functionality is never* added to them but is added to the tools that work with the ObjectBase or within the intelligence of the ObjectBase itself.

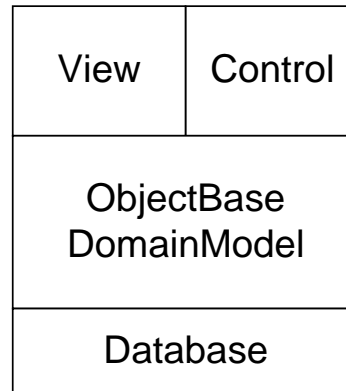
Non-MONDO Applications and Frameworks

MONDO's features are limited compared to all the needs and possibilities with information processing applications. Other needs could be:

15. UI Presentation and Interaction
16. Database Storage
17. Sophisticated Queries
18. Distributed Execution
19. Physical system interconnection

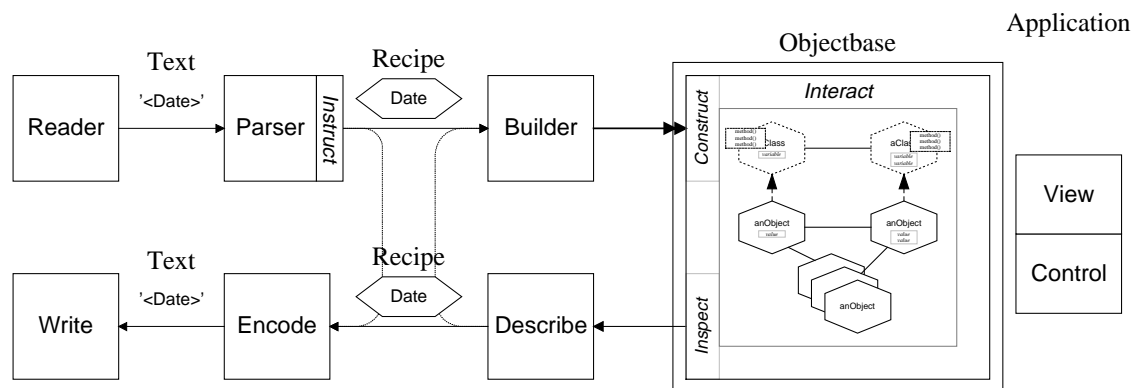
* The ObjectBuilder lives on the border. It has well-define responsibilities but can also become smarter about its recipe building to support more sophisticated DomainModels and recipes. It is still focused on building the ObjectBase.

All of these other pieces of functionality can come through the ObjectBase and DomainModel, so applications built using MONDO will be able to work with them. For example, the application could use the functionality of an ObjectServer to do OQL-based queries. Or it could use a domain presentation framework to view and control* the domain model with a sophisticated UI. As long as the interchange is in the ObjectBase (and the frameworks work well cooperatively), any number of different pieces can be integrated together



Summary

MONDO's overall architecture has the following major components and interactions.



Each of these component has clear responsibilities, simple interfaces, and can be individually reused. The flow of information can go in several ways but is focused toward only three goals: building domain objects, building recipes, and producing text files for recipes.

2.3 Functionality Levels

MONDO's core architecture is very general: It defines concepts, components, interfaces, and responsibilities. This core architecture can be the foundation of many different types of systems and the architecture can be implemented in any programming language. It is very flexible but, by itself, it does not provide much functionality.

MONDO defines **functionality levels** that extend this core architecture to support the needs of many different types of applications. **Level-0** (L0) is the implementation of the core architecture itself and all the language independent functionality. Most of L0 is implemented in Interfaces, Abstract Classes, or it can even be "implemented" through documentation. L0 also provides test functionality that can help applications test whether they work properly with the MONDO interfaces. **Level-1** (L1) provides integration with the functionality available in the host language environment. This includes supporting Java Beans, Smalltalk's reflection, or C++ RTTI. L1 only cares about very general functionality that many applications will need, but which are only applicable within a given system environment (e.g. programming language). **Level-2** (L2) provides component services on top of Level-1.

* See [Gamma+HJV 95], [Buschmann+MRSS 96], or [Howard 95] for information about the Model, View, and Controller Pattern and its extensions.

Because MONDO is still a young project, the functionality levels have not yet been fully and formally specified.

2.4 Summary

MONDO has core concepts of DomainObject and Recipes. The MONDO architecture support building, interacting with, and transforming these core elements using major subsystems of the ObjectBase, the ObjectBuilder, and the ObjectEncoder. Within each of these major subsystems are components which more finely organize and support the processing of information. These internal components support simple, focused responsibilities and the ability to interconnect and reuse components easily.

The MONDO concepts and architecture are very simple but also very powerful. The following chapters will discuss the design details of the major components and, by so doing, provide a full picture of what the MONDO vision and concepts are.