Core White Paper

What is Core?

Core is a high level declarative object oriented systems development environment. It is, among other things, the platform for systems supporting planning, development, procurement and delivery of all equipment to the Swedish Armed Forces. Core is developed and maintained by Genicore AB, Göteborg, Sweden.

Purpose of Core

Core is intended to reduce cost and increase quality of corporate information systems. It includes features such as complete information integration, client-server architecture, parallel processing, real-time information distribution, user friendliness, high reliability and military grade security.

Proven advantage

It has been shown that when replacing or comparing systems based on other modern commercial technologies with systems based on Core technology, development and maintenance effort is reduced by 80–90 %. At the same time, system integration, functionality and user friendliness is greatly improved.

Core development environment

Core is based on the idea of a business information model, not to be confused with the traditional data model. The *business information model* includes all business data definitions and processing definitions of the system. The information model is defined in the Core model tool, *CoreBuilder*. CoreBuilder generates C++ code, SQL code and database scripts for the target environment directly from the information model.

A *graphical user interface* is built with the *Core-UI* environment. This graphical environment connects to metadata of the information model and contains high level parameter controlled components. The user-interface is assembled, not programmed, with these components, by connecting them to each other and to the information model.

CoreBuilder and Core-UI both generate C++ code. Together with some libraries, this code is compiled and linked to produce the runtime system.

A Core system is usually divided into several applications, which run independently. But all applications include the same information model and access the same server. Thus all information is automatically integrated and synchronized between all applications.

The *data import and export* function uses an integration framework called CoreCom. CoreCom is fully integrated with the Core business model, but can also be used independently to integrate any kind of legacy systems. Data import/export and data conversions are described using the power of XML/XSLT language. Import and export definitions are installed by importing them into the system. Changes and extensions can be made in run-time.

CoreWeb is a framework for rapid construction of full-featured WEB applications connected to the Core business model. CoreWeb is completely built around WEB standards, such as XSLT, JavaScript and html. CoreWeb applications are described in XML and can be modified and extended at runtime.

A Report generation and alternate UI system using MS-WORD as UI agent is also available. It is defined in XML and XSLT. With this UI system, it is possible to describe arbitrary documents, which are generated from information in the system. It is also possible to open "live" documents where the user can make changes to the document in MS-Word, which then updates the corresponding information in the system.

Core Meta Model

The structure of the Core information model is defined by a meta-model, which can be regarded as an augmented abstract syntax tree, AST. The meta-model is also described and developed in Core.

The Core information model is object oriented. Its class structure is based on single inheritance. Class members are called *attributes*.

Core supports a *declarative* programming structure. Imperative structure of programming is not possible in Core. Instead, the assignment of the value of an attribute is defined by an expression inside the attribute. To traverse structures of information, a recursive and polymorphic approach is used.

Expressions are defined in C++ syntax with some extensions. The expressions themselves can be imperative, but they should not have any side effects.

Attributes of classes are not just simple placeholders for value of a certain type and an assignment expression. Attributes also maintain several internal runtime states, such as current read/write permissions. The value of these states can themselves be defined declaratively, by non side effect expressions. The Boolean expressions, such as the expressions defining read/write permission, are often referred to as rules.

Attributes come in three variations, *value attributes*, *state attributes* and *relation attributes*.

Value attributes work as can be expected. They have a value, and optionally also one or several expressions. Value attributes can be of any type, including Class and ClassSet.

State attributes add the feature of implementing a state-transition graph. Transitions can be triggered from outside the system or by their own defined expressions. Externally triggered transitions can be validated by expressions, with an automatic backtracking feature, giving the reason for a validation fail.

Relation attributes come in two flavors, single relation and multiple relation (set). They are typed and contain references to objects. Relation attributes are always defined in pairs, and the runtime environment will ensure the integrity of these dual reference pairs.

Datatypes in Core are Class (any class defined in the information model, including the base class ‘Object’), ClassSet (one type for each class of the system) and Value (Text, Float, Integer, Date, Time, Boolean etc).

Datatypes define functionality of applicable operators including typecasting operators and value-states. Examples of value-states are ‘undefined’ and ‘not valid’. Types can be parametric and contain additional static information, such as range limitations and enumeration.

Runtime features

The following are intrinsic features of all systems built with Core. No specific programming is required to include these features. The Core libraries and the generated code will automatically include all functionality listed below.

Client server distributed processing: All data processing is performed on the clients. The server stores data and coordinates data sharing and distribution.

User identity, user login and permissions system: Core maintains definitions of user profiles and controls read/write permission on attribute level.

Real-time: Information is distributed in real-time among the clients – sometimes there are thousands of them. When a piece of information is changed, it is distributed among all the clients currently holding copies of this information.

Persistence: All objects and attributes of the information model are persistent by default, although it is possible to exclude specific attributes from persistence.

Long transactions: Long transactions are allowed and possible due to a pessimistic locking system. This means that the user can work on any set of data for an extended time. Ongoing changes by one user are dynamically protected from being overwritten by others.

Pessimistic locking system with automatic trace of dependencies: The system will lock the scope of a change before it can take place. The locking mechanism is transparent to the users, unless locking fails, in which case the user is informed that changes can not be made at this time.

Complete WEB-service: All information in the information model is by default accessible for both read and write through a generic WEB service. Data requests and updates are described with XML schemas.

Versioning system: A versioning system keeps track of all changes of information. It is possible, at any time, to go back to any previous time of the system and inspect data and data changes. It is possible to step back and forth for each data change of any selected object.

Dataflow execution: Due to the declarative style of the system definition and to the fact that all processing is scheduled by a data dependency mechanism, system execution is equivalent to a data flow system.

Usage

The FMV-Core system – 15 applications, 3500 users, more than 200 information classes in an up to 11 levels deep hierarchy, 1500 value and state attributes, 400 relation attributes, 30 GByte of mostly textual and numerical data – is equivalent to a traditional system with about 1000 database tables and 3500 transactions.

Core is also used to create new versions of CoreBuilder, the information modeling tool.