ORIMOS

SONARIS/Framework

Overview and Concepts

Version 1.4.3

ORIMOS S.A.

Innere Gueterstrasse 4
6304 Zug (Switzerland)

Phone +41-41-725-3570
Fax +41-41-725-3579

Web www.ORIMOS.com
Email info@ORIMOS.com

ORIMOS Group
Berlin · Frankfurt · London · Zurich

Contents

Preface

This guide provides a high level introduction to the fundamental design concepts of the **SONARIS/Framework**. Chapter 1 of this guide introduces the **SONARIS** architecture. Chapter 2 presents a brief summary of the features and services provided by the **SONARIS/Framework** environment. At appropriate points the reader is referred to other guides in the documentation set which cover the topics introduced here in more detail. Please also refer to the glossary which appears at the rear of all the guides in the documentation set for descriptions of specific terms used.

1 Architecture

1.1 Introduction

Up until now, achieving optimum performance using standard object-oriented approaches and languages has required the compilation of program code into fixed executables or libraries which, once deployed, are inflexible and difficult to extend. Conversely, greater flexibility has previously only been available through the use of interpreted environments at the cost of performance.

Whereas a conventional object-oriented design consists of objects that have structure and relationships fixed at compile time, **SONARIS/Framework** components have uniform interfaces providing normalised data flow. With **SONARIS/Framework**, compiled objects are connected together dynamically, providing both performance and flexibility. This results in the powerful ability to add your own new system components at runtime.

SONARIS/Framework is designed to enable the decomposition of complex objects into simple components called *SONARIS Application Objects (SAOs)*. Structures are created by combining these simple components to build flexible real world objects, which can be extended at runtime. This enables business solutions to evolve to address new requirements while ensuring maximum system availability. **SONARIS/Framework** conforms to a server/client model whereby heavy processing is performed on powerful server machines and the results sent to a lightweight interface running on client machines.

It is this architectural simplicity that gives it its strength and flexibility. Each SAO receives inputs and produces an output without having to concern itself with where the inputs come from or where the output goes. This decoupling of an SAO from its data providers and consumers strongly distinguishes **SONARIS/Framework** from other object-oriented development tools; component behaviour and data flow between components can be modified without additional coding, linking or compilation. The standard object-oriented approach would require at least one iteration of these three steps before the required changes could take effect.

1.2 SONARIS Application Objects

SONARIS Application Objects (SAOs) are the basic building blocks of **SONARIS/Framework**. Each SAO instance takes a number of *inputs* and applies some processing to these inputs in order to derive a single *output* value and status. The SAO code that performs the processing is obtained from a library that can be loaded at run-time i.e. a Dynamic Link Library (DLL) on Windows and a shared library on Unix. The term *SAO Library* is used to collectively refer to such libraries. The various types of SAOs available from such libraries are distinguished using a two part *Functional Type* number which identifies the SAO library which supplies the SAO code and an index number for each SAO within that library. For convenience a name and category are associated with each SAO type within the library and each library itself has a name.

Every instance of one of these SAO types used within the Framework is assigned a unique number and name. These instances are then be grouped together in a *Hierarchy* which expresses a parent/child relationship between the SAOs. Every SAO in the hierarchy (with the exception of the root SAO) is owned by a parent. A real-world object (e.g. a financial instrument) may be represented by such a hierarchy of SAOs, the values that belong to the real-world object being held by the SAOs of the hierarchy.

Figure 1.1 shows the key features of an instance of an SAO including its name, id, functional type name, and current value and status.

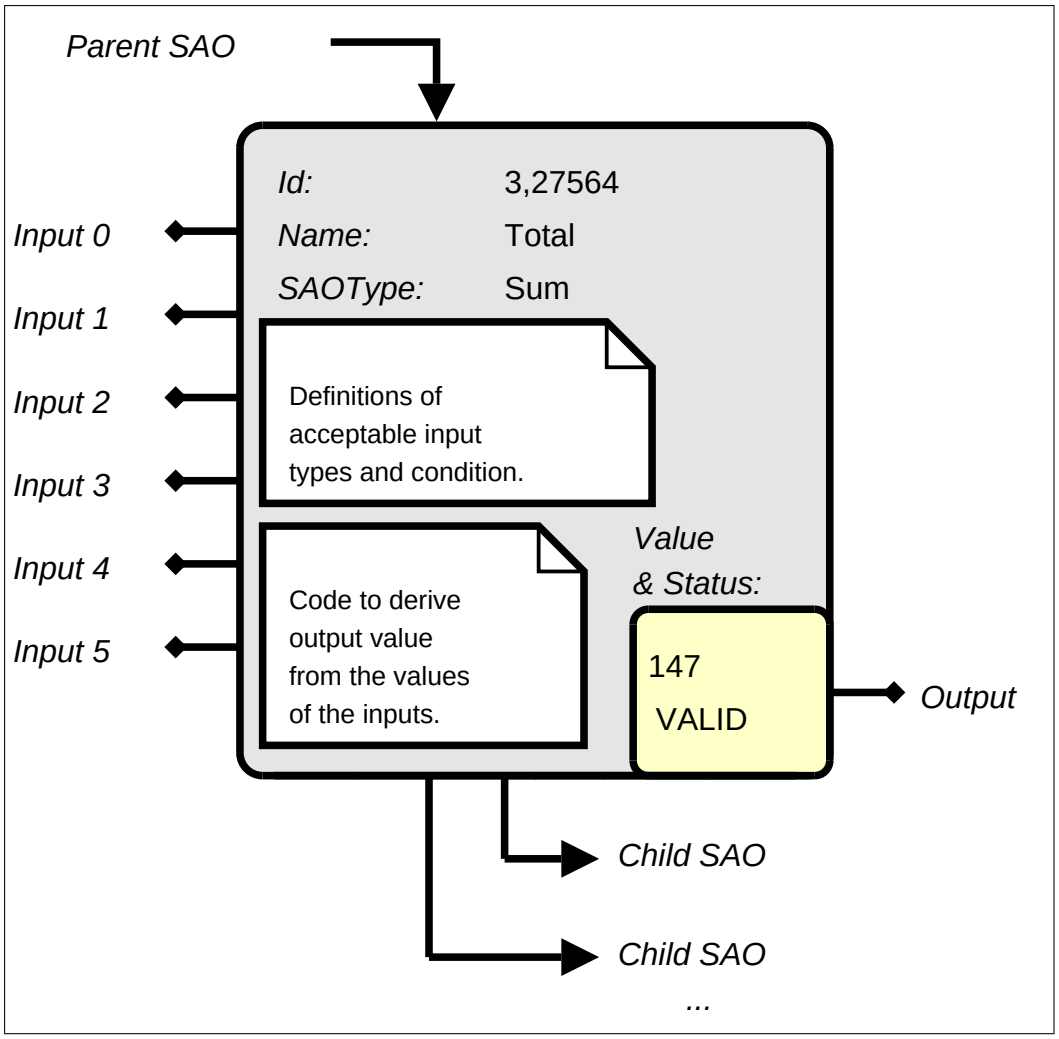

Figure 1.1: A Typical SAO

The output value of each SAO instance is available as an input to zero, one or more other SAOs. These SAOs are said to be *dependants* of the SAO providing its output value. The network which describes the flow of information between the inputs and outputs of SAOs is called the *dependency graph* and it is completely unrelated to the hierarchical location of the SAOs involved.

1.2.1 An Example

Figure 1.2 shows some SAOs used to perform a simple foreign exchange calculation which illustrates the distinction between the hierarchical location of SAOs and their dependency relationships. The SAOs involved are of the following functional types:

basicDouble An SAO with no inputs which holds a double value.

doubleDivide An SAO with two inputs which stores a double value which it calculates to be the result of dividing the value of its first input by the value of its second input.

mean An SAO with unlimited inputs which stores a double value which it calculates to be the mean of the values of its inputs.

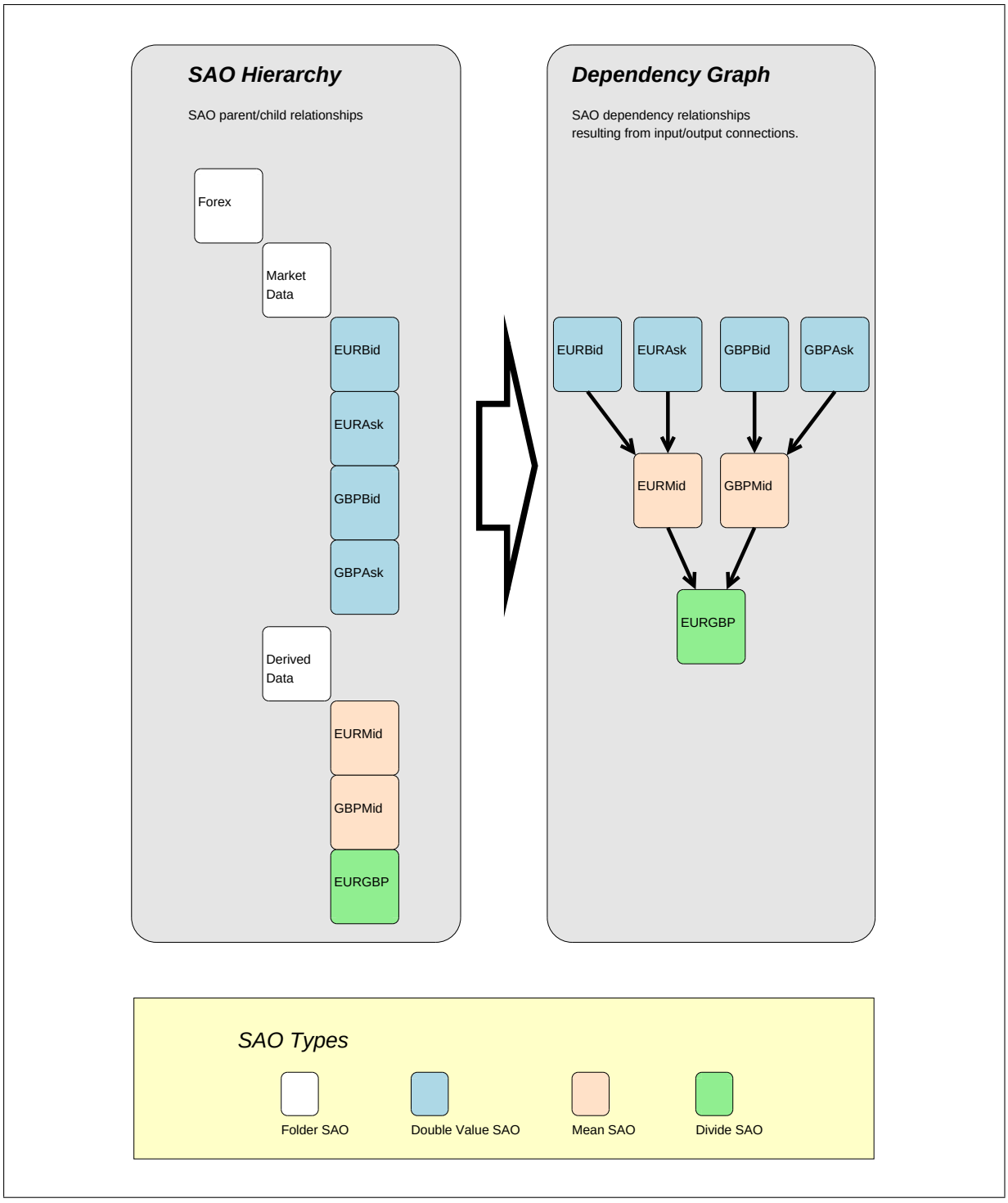

Figure 1.2: An SAO Hierarchy and Dependency Graph

folder An SAO which has no value, no inputs and no outputs and is used purely as a place holder in the SAO hierarchy.

The result of the calculation is held by the `EURGBP` SAO which is an instance of functional type *doubleDivide* and which is located in the hierarchy as a child of the `Derived Data` folder SAO which is itself a child of the `Forex` folder SAO. The `EURGBP` SAO calculates its value to be the quotient of its two inputs which are the values of the `EURMid` and `GBPMid` SAOs which also

appear under the `Derived Data` folder SAO. Both the `EURMid` and `GBPMid` are instances of the *mean* SAO type, and these calculate their values as the mean of their inputs which are the bid and ask values for the two currencies with respect to a third currency. These are stored in the `EURBid`, `EURAsk`, `GBPBid` and `GBPAsk` SAOs located under the `Market Data` folder.

Hence any change in the values of the SAOs which store the bid and ask values for the currencies will prompt a recalculation of the related mid values and finally a recalculation of the exchange rate.

It is important to note from this example that the hierarchical locations of the SAO instances are in no way related to their dependency relationships. It would be equally valid (though perhaps less logical) to arrange the SAO instances such that they all had a common parent. Alternatively, separate folders might be introduced to parent the market values for the currencies. Either way the dependency relationships between the SAOs would be the same.

1.3 Containers and Processes

As outlined above, SAOs can be connected together in a certain way to achieve a specific task. Such an assemblage of SAOs is referred to as an *Application Service*. Within **SONARIS/Framework** groups of SAOs are managed by one or more instances of objects called *containers* which perform the task of maintaining the hierarchy and dependency relationships between the SAOs. Each container instance holds a portion of the SAO hierarchy which represents the application service and provides services to the SAOs that it contains. In particular, the container provides the thread which executes the code of the SAOs that it hosts. The container is responsible for notifying each SAO as appropriate when the value or status of any of its input SAOs changes so that an updated value can be calculated. Such a sequence of notifications to SAOs is sometimes called a *ripple*. The container is also responsible for delivering new values from outside to the SAOs it hosts and for passing on updated values in those SAOs to *client* programs or other containers.

Container objects are created by instances of **SONARIS/Framework** *processes*. Each process may be configured to host one or more containers which hold SAOs from the same application service. Scalability therefore comes from two sources:

- The ability to split an application service across two or more processes. This means that an application can be split across several machines.
- Each container having its own thread means that a process containing several containers can make full use of a multi-processor machine.

The ability to distribute an application service across multiple processes and machines also brings the benefits of greater resilience since the failure of one machine does not result in the failure of the entire application service. It also allows the integration of resources available on different machine architectures and operating systems into a single application service.

1.3.1 Communication

The exchange of data between containers and clients or containers and each other is achieved through the use of the **SONARIS/Messaging** *Message Orientated Middleware (MOM)* which is an integral part of **SONARIS/Framework**. Any SAO hosted by a container is available for subscription by clients and also by other containers.

Client programs may access the SAOs presented by an application service without any concept or knowledge of the configuration or location of the containers and processes which host the application service. At the client side, an API is layered on the MOM. This API gives clients access to the values of any of the SAOs that form part of the application service to which the client is connected. All routing of subscriptions and the updates which result from them is handled transparently by the client API and the MOM. Client programs may establish subscriptions to individual SAOs or portions of the SAO hierarchy simply by specifying the hierarchical location of the SAOs required. They then automatically receive updates when changes occur to the value or status of the SAOs or to the structure of the hierarchy. For more information regarding the client API refer to the *SONARIS/Framework - C++ Client Programming Guide*, *SONARIS/Framework - Java Client Programming Guide*, *SONARIS/Framework - C# Client Programming Guide* or *SONARIS/Framework - COM Client Programming Guide* as appropriate.

1.3.2 Configuration

The configuration information for an application service is persisted in a common database abstracted behind each container. Details about the SAO instances and their hierarchies and dependencies are stored in this database. In addition, the current values of SAOs may be stored and updated automatically when they change. The database schema is simple and is accessed via a normalised interface which is also implemented as a library, so different databases may be used to store this configuration information without changing the application whose configuration it supplies. Refer to the *SONARIS/Framework - Release Notes* for details of the database systems supported by **SONARIS/Framework**.

1.3.3 Communicating with Other Systems

As mentioned previously, one of the services provided by the container to the SAOs it hosts is the delivery of data from external sources. All data arriving at the container is passed through a queue before being routed to the SAO which is its intended recipient. The SAO is then free, depending on its functional type, to process the new data as appropriate and possibly adopt a new value. If the value or status of the SAO changes then all other SAOs which have a dependency on it will be marked by the container to be included in the next ripple and will have their own values updated accordingly.

Many **SONARIS/Framework** applications need to receive data from some external source (e.g. Market Data systems, databases etc.). To enable this the container makes its input queue available to external systems via an API (the *External System* API) and a standard SAO library is supplied which maps data structures presented by external systems into a hierarchy of SAOs. These SAOs issue requests to the external system via the the API and receive resulting data delivered back to them via the container input queue.

1.3.4 An Example

The diagram in Figure 1.3 illustrates how the components of **SONARIS/Framework** work together to provide a flexible and scalable solution. It shows two **SONARIS/Framework** processes with one running two containers and the other just one, and a few representative SAOs. It also shows three **SONARIS/Framework** clients.

Points to note in the **SONARIS/Framework** processes are:

- SAO hierarchy relationships in the container are shown as a vertical sequence of the coloured (or grey-scale) squares and SAO dependency (i.e. data flow) relationships are shown with solid lines.
- A number of External Data Sources and SAO Libraries can be used by a single Process at any one time.
- A number of External Data Sources and External Gateways may be used by the Processes to exchange data with External Systems.
- All inputs to SAOs are channelled through a queue that belongs to the container. This is true even if the input has come from another SAO in a different container in the same process or from a process on another machine.
- The colour (or grey scale when printed non-colour) of the SAOs indicates which SAO library they come from. This shows that SAOs from several independent sources can be joined together to produce the solution to the problem.
- A single input can be supplied to more than one SAO.
- The basic SAO can take inputs directly from the container input queue and from other SAOs (connections from other containers are implemented via the container input queue).

Points to note again in the **SONARIS/Framework** clients are:

- A client can receive information from an arbitrary selection of SAOs.
- A client has no concept of the containers or processes which host the different areas of the hierarchy, it simply has a view of the entire application hierarchy composed of SAOs.

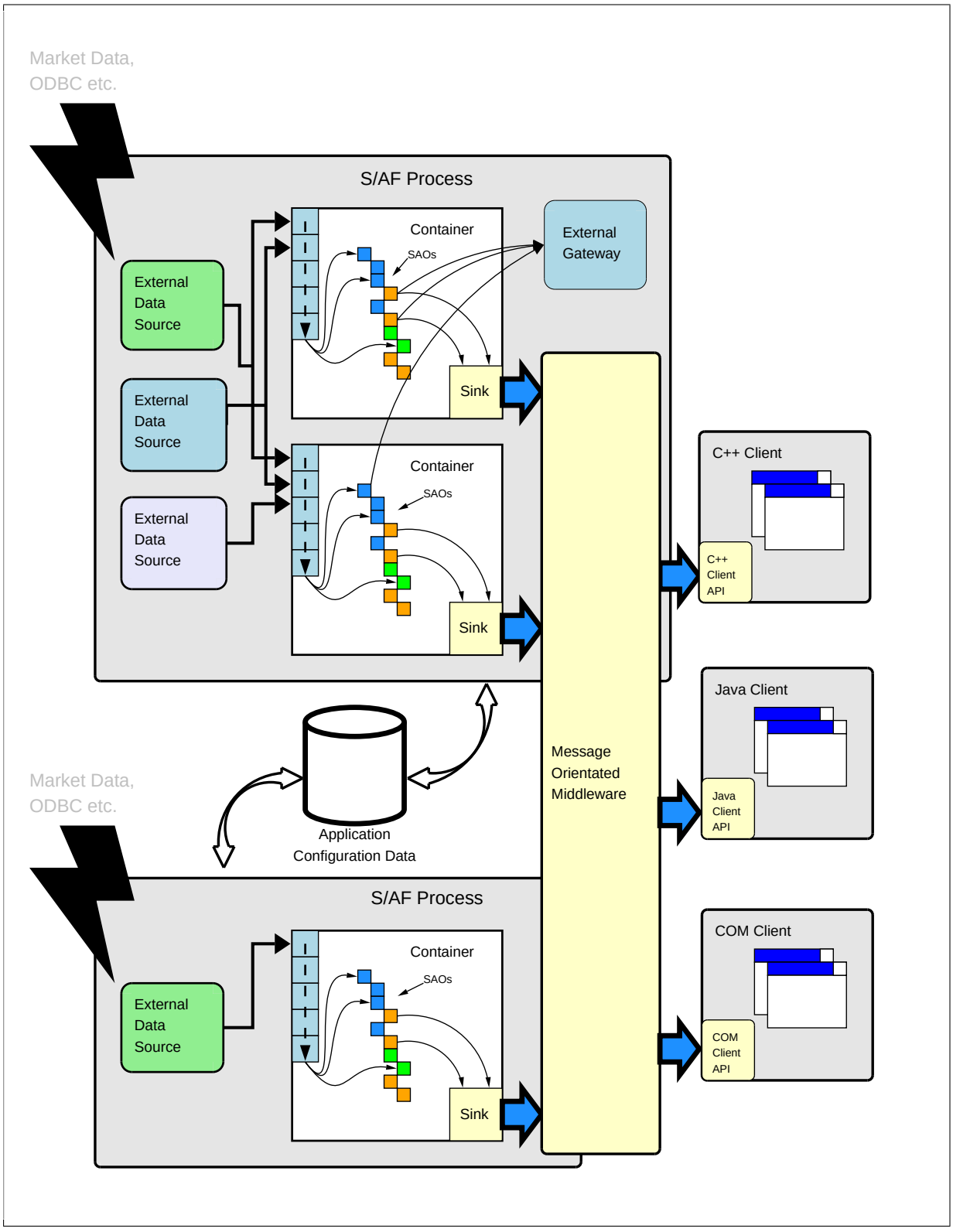

Figure 1.3: The **SONARIS/Framework** Architecture

2 Features

2.1 Programming and Administration

SONARIS/Framework is supplied with an extensive collection of SAOs which can perform such tasks as common calculations, text manipulation, logical operations and timing functions, and provide access to external data sources. Other libraries are also available which provide SAOs for more sophisticated financial calculations, manipulation and monitoring of SAO hierarchies etc. For a full listing of all the SAOs supplied with **SONARIS/Framework** refer to the *SONARIS/Framework - SAO Catalogue*.

A powerful user interface, **SONARIS Studio**, is supplied with **SONARIS/Framework** which gives a graphical representation of the SAO hierarchy which forms an application service and gives access to SAO libraries to allow the hierarchy to be monitored and extended by a suitably authenticated administrator. With this program it is possible to:

- Create new application services.
- Monitor and extend existing application services.
- Load details of new SAO libraries for use with application services.
- Dynamically assign portions of the SAO hierarchy to different machines, processes and containers.
- Administer user access to application services.

A full description of the usage and features of **SONARIS Studio** is available via its on-line help. For an introduction to creating and modifying application services refer to the *SONARIS/Framework - Application Programming Guide*.

2.2 Installation and Maintenance

A detailed description of the installation procedure for **SONARIS/Framework** appears in the *SONARIS/Framework - Installation and Maintenance Guide* along with reference information for all executables supplied and a more detailed description of the following topics:

- Running application services.
- The function of executables and relationship between them.
- Logging and monitoring.
- Database configuration.

2.3 Custom SAOs

Beyond providing ready-to-use SAOs, **SONARIS/Framework** also gives developers the opportunity to encapsulate their own code as SAOs which can then be integrated into application services along side the standard SAOs. The encapsulation is performed by implementing a simple API which allows the container to interact with instances of the SAO. The key methods of this

API allow the container to supply updates to the SAO from the container's input queue and to inform the SAO if an SAO on which it depends has changed value or status to allow it to participate in ripples. For more information about creating custom SAOs refer to *SONARIS/Framework - Custom Component Programming Guide*.

Glossary

Active Interface
An interface to a *SONARIS/Framework* programming object for which a standard implementation is available. Instances of objects which provide this implementation are supplied by the *SONARIS/Framework Runtime Environment* and can be queried and manipulated via their interface definition. See also *Listener Interface*.

Application
In *SONARIS/Framework* terms, a complete solution composed of an *application service* which processes and publishes data using *SAOs*, and *client* programs capable of accessing the data held by those SAOs.

Application Service
In *SONARIS/Framework* terms, a group of *containers* hosted by *processes* using *SAOs* to implement a service capable of supplying the data held by those SAOs to *client* programs.

Child
In terms of *SAO hierarchies*, an *SAO* which is parented by another SAO within a hierarchy is said to be a *child* of that SAO.

Client
In *SONARIS/Framework* terms, a program capable of subscribing to the data held by *SAOs* which form part of an *application service*.

Configuration Database
A database (usually hosted by some RDBMS system) which acts as a store for configuration information that describes *hierarchies* of *SAOs*, the properties of those SAOs (value, history, log level etc.), the *connections* between them, and by which *container* they are hosted as part of an *application service* (refer to the *SONARIS/Framework - Release Notes* for details of the RDBMS systems supported).

Connection
A *dependency* relationship between two *SAOs* which specifies that the value of one of the SAOs is dependant on the value of the other, and that changes in the value of the former will be reflected in the value of the later, or that the later will take some action based on the change of the former.

Container
An object created by a `saf` *process* responsible for hosting a *hierarchy* of *SAOs* and servicing them with data supplied from other containers, *clients* or *External System Retrievers*. A container uses a single execution thread to deliver *updates* to SAOs and then, by means of a *ripple*, propagate resultant changes to other *dependant* SAOs and supply updates to other containers or clients which hold subscriptions to the affected SAOs.

Cycle
A pattern in the *dependency graph* of *connections* between *SAOs* which revisits an SAO during a single *ripple*. The most direct form of a cycle occurs when an SAO uses its own *output* value as one of its own inputs. Cycles in the dependency graph can be used to give controlled *feedback* effects, but should be used with care, as they can yield results which are difficult to interpret.

Data Source
In *SONARIS/Framework* terms, a source of configuration data which can be read in order to initialise a SONARIS/Framework *application service* (usually a relational database). See *Configuration Database*.

Dependant
In *SONARIS/Framework* terms, an *SAO* whose value is established and maintained based on a *connection* which uses the *output* value of another SAO as one of its *inputs* is said to be dependant on that other SAO.

Dependency Graph
The set of *dependency* relationships established as a result of forming *connections* between *SAOs*.

ESR
See *External System Retriever*.

ESS
See *External System Sender*.

Export
The process of creating an XML description of a *hierarchy* of *SAOs* which can then be used to create a copy of that hierarchy in another location in, possibly, another *application service*. See also *Import File*.

External System Retriever
An implementation of the **IExternalSystemRetriever** interface (defined in `isafesr.h`) which wraps third party software provided by an external information supplier allowing data to be passed to individual SAOs within a container. An implementation of an ESR will typically map the information available from the external supplier into a hierarchical namespace which corresponds to hierarchies of SAOs that can be created within application containers. If the supplier being wrapped also supports the return of data for external publishing or contribution a combined IExternalSystemSender interface may also be supplied.

External System Sender
An implementation of the **IExternalSystemSender** interface (defined in `isafess.h`) which wraps third party software provided by an external information publisher allowing data to be sent from individual SAOs within a container for publication by the external system. Like an *ESR*, an implementation of an ESS will typically map the third party structures via which data can be published into a hierarchical namespace which corresponds to hierarchies of SAOs that can be created within application containers.

Feedback
A condition arising from a *cycle* in the *dependency graph* where the value of an *SAO* is set during a *ripple* but not used by one of its *dependants* until a later ripple.

Hierarchy
A complete tree, or branch of a greater tree, of *SAOs* hosted by one or more *containers*. The complete hierarchy representing an *application* is usually composed of several individual hierarchies hosted by separate containers distributed across numerous *machines* and *processes* and joined together at *mount points*.

Import File
An XML file containing a description of a *hierarchy* of *SAOs*. Generally, such files also contain information about *connections* between SAOs in the file as well as their values. Import files can be used to create complex hierarchies of connected SAOs from a single command.

Input
Part of the *definition* of an *SAO* which specifies the end point of a *connection* formed from some other SAO to this in order that this SAO be able to use the *output* value of that SAO in establishing and maintaining its own value.

LCS
See *Licence/Configuration Server*.

Licence/Configuration Server
A *node* within a *partition* configured to manage configuration, licencing and connection topology for the other nodes within the partition.

Listener Interface
An interface to a *SONARIS/Framework* programming object which must be implemented by a program or component wishing to communicate with *SONARIS/Framework Runtime Environment*. See also *Active Interface*.

Machine
In *SONARIS/Framework* terms, a host on which *processes* hosting *containers* are run to provide all or part of an *application service.*

Mount Point
A defined location within a *hierarchy* of *SAOs* hosted by a *container* at which another hierarchy, hosted by another container is rooted.

Node
A machine on which *SONARIS/Messaging* is installed and which forms messaging connections to other nodes within the same *partition*.

Output
The current value of an *SAO* available to any *client* which holds a subscription to it, or to any other SAO which uses it as one of its *inputs*.

Parent In terms of *SONARIS/Framework hierarchies*, an *SAO* which has one or more *child* SAOs is said to act as the *parent* for those SAOs.

Partition
A group of *nodes* one of which is configured to act as *LCS* for the others.

Persistent SAO
An instance of an *SAO* whose details (including Id, name, *hierarchical* location, ownership details, *input* details etc.) and (optionally) value, are stored in the *configuration database* for the *application* such that the SAO will be available every time the *container* which hosts it is run.

Process
In *SONARIS/Framework* terms, an individual invocation of the `saf` program which creates and manages one or more *containers* to provide part or all of an *application* service.

Proxy SAO
A subset of the full functionality of an *SAO* used to remotely represent an SAO in either a *client* program or a *container* other than the one which hosts it.

Ripple
A sequence of calls by a *container* to methods of a subset of the *SAOs* managed by that container to ensure that the values held by those SAOs is brought up to date with respect to one another, and to pass on any value changes which have occurred to *client* programs which hold a subscription to those SAOs. A ripple is usually performed as a result of new data being supplied as *updates* to one or more SAOs from some external source (another container, a client, an *ESR* etc.) to allow any *dependant* SAOs to modify their values accordingly.

SAO
See *SONARIS Application Object*.

SAO Definition
A formal description of an *SAO* that provides a general description of the SAO, defines the type of data it is capable of storing, and specifies the number, types, usage conditions, and general descriptions of its *inputs*. Such definitions are supplied with every *SAO library* as XML text, and must be loaded by every *application* which makes use of that library to verify appropriate usage of instances of the SAOs within the application.

SAO Library
A library of *SAOs* supplied in two parts; a dynamic linked library binary file specific to a particular computer architecture, and an architecture and operating system neutral *SAO library definition* file which formally describes the SAOs provided by the library using XML.

Sibling
In terms of *SONARIS/Framework hierarchies*, an *SAO* whose *parent* also acts as a parent for one or more other SAOs is said to be a *sibling* of those SAOs.

SONARIS/Framework Runtime Environment The core features and services of the *SONARIS/Framework* system available to client programs and SAOs.

SONARIS/Framework
Component based, distributed development framework for implementing real-time calculation and data processing solutions.

SONARIS Application Object
The most basic component of a *SONARIS/Framework application*. SAO instances are hosted by *containers* and arranged in *hierarchies*. The implementations of SAOs are supplied by *SAO libraries*.

SONARIS/Messaging
The underlying messaging system used by *SONARIS/Framework* for the exchange of data between SONARIS/Framework *containers* and *clients*.

Template

A branch of a *hierarchy* of *SAOs* which serves as a master copy to be replicated at other points in the hierarchy.

Transient SAO

An instance of an *SAO* which, unlike a *persistent SAO*, is only available for the life time of the of the *container* which hosts it. The Id assigned to a transient SAO will not be used again by any other SAO.

Update

A delivery of a new value to an *SAO* (usually from outside of the current *container*) or from a container to a *client*.