3.1 ORM Protocol Layer And Upcall Interfaces

This section was generated from *<stdin>* by CDOC on Sun Jan 29 17:00:50 1995.

ORM Application Context

Application Server Capsules may serve different kind of requests and therefor may have multiple domains of objects to be managed listening on multiple ports. Following the ORM model, this may result in multiple parallel independant trees.

The ORM parser supports this by maintaining an application context, which has to be passed to the protocol layer to handle a request (there is also an opaque *call-context*, which may be passed to the protocol layer, but this isn't interpreted by the ssl).

The application context contains beside (an opaque pointer) to the (virtual) root of the virtual tree, mainly a list of tree/application specific function pointers. Before the first request can be passed on to the ORM protocol layer, this context has to be established with the ORM SSL via a call to ORM_ContextInitialize.

Accordingly there exists a function to inform ORM that this application context is not needed anymore (release).

The following lists the function prototype definitions for actual functions to be provided, when establishing a context.

Note: Some functions are defined to return pointers to character strings (ORM_String). If the ORM protocol handler is used it is guaranteed, that the same function will not be called, before the string is copied or otherwise not needed anymore. This allows the use of a single private string buffers per function, if necessary.

3.1.1 Authentication

The following list of functions are included to enable an application to maintain its own authenticated context. The ORM protocol just allows to forward some authentication related information from the client to the server (WHO...). This is passed on to the application layer as is, if encountered by the parser. The actual meaning of this data is application and user interface dependant.

3.1.2 Function Type ORM_AuthenticateFunc

Performs any necessary authentication or preparation of authentication structures. Usually, the authentication information is used to setup some context in the call-context, which is passed to the node/handle layer upcalls. It is up to the application layer to free/clear such context after return from the protocol layer.

Declaration:

```
typedef ORM_Status (*ORM_AuthenticateFunc) (
                ORM_AppCallContextDef    callcontext,/* in*/
                ORM_String               authstring  /* in */
```

Fields:

callcontext	An opaque pointer to any kind of context, the caller has established.This passed to the node and handle layer.
authstring	The string, the client passed in his request, if any. Usually uid:passwd
status	ORM_ENoError: if successfull, ORM_EPermissionDenied, if authentication unknown.

3.1.3 ORM_AuthFuncDef

This structure is used to pass the Authentication function to ORM_ContextInitialize

Declaration:

```
typedef struct ORM_AuthFuncTag {
    ORM_AuthenticateFunc        auth;
} ORM_AuthFuncDef;
```

3.1.4 Virtual Node & Tree Function Types

The following list of functions (function types) are used to access the virtual tree of components, attributes and linked objects. They usually don't deal with application specific data.

3.1.5 Function Type ORM_NodeLookUpFunc

This is the central function for the traversal of *the* tree .

Returns an opaque pointer to a virtual node, which may subsequently be called to retrieve properties or children of specific types.

Declaration:

```
typedef ORM_Status (*ORM_NodeLookUpFunc ) (
        ORM_AppCallContextDef    callcontext,/* in */
        ORM_AppNodeDef           root,       /* in */
        ORM_String               pathname,   /* in */
        ORM_AppNodeDef           *node,      /* out */
        ORM_NodeTypeDef          *nd_type    /* out */
    );
```

Fields:

callcontext	is an opaque pointer to the application specific call context provided with the Do_Request function.
root	Opaque Pointer to root of virtual tree. This may be NULL, and is taken from the application context.
pathname	is a / separated list of component names optionally preceded by the name of the object (e.g. if the first component matches the roots object name, strip it, else take the first component to be a child under the applications root). Support for *un*x* style directory navigation . and .. is highly recommended/required. A pathname of .. applied to the root with request type *Object* should return the root name and the actual servers link address (NOR)
node	The opaque node pointer, if found
nd_type	The ORM_NodeType of the node found
return	ORM_ENoError in case of success, or any other ORM error in case of failure.

3.1.6 Function Type ORM_NodeChildNextFunc

Used to subsequently scan the children of a single parent. Returns the next child of type *type* of parent *parent*, which logically follows the child returned by the previous call to *NodeChildNext()*, now passed in as *lastchild*. E.g. If *lastchild* is set to *NULL* the logically first child of this parent is requested. If there are no children (of the requested type), then NULL must be returned with ORM_Status set to ORM_NoError.

Declaration:

```
typedef ORM_Status (*ORM_NodeChildNextFunc ) (
    ORM_AppCallContextDef   callcontext,/* in */
    ORM_AppNodeDef          parent,     /* in */
    ORM_AppNodeDef          lastchild,  /* in */
    ORM_NodeTypeDef         type,       /* in */
    ORM_AppNodeDef          *child,     /* out */
    ORM_String              *name       /* out */
    );
```

Fields:

callcontext	is an opaque pointer to the application specific call context provided with the Do_Request function.
parent	Opaque pointer to the virtual parent node.
lastchild	Opaque pointer to the last child returned by a call to this function (in this request), or NULL to request the first child.
type	The type of entity, which is requested (ORM_ObjectType, ORM_ComponentType, ORM_AttributeType or ORM_AnyType).

node	Pointer where to store the reference to the node found
name	Pointer to name of node found.
returns	status value. Possible status values, see below!

3.1.7 Function Type ORM_NodeChildByNameFunc

The little sister of ORM_NodeLookUp. Looks for a child with name *childname* directly under the given parent *parent*. This function is primarily used within the processing of Set- Attribute requests. If there is no child with this name, return NULL and an error status (see below)

Declaration:

```
typedef ORM_Status (*ORM_NodeChildByNameFunc ) (
        ORM_AppCallContextDef   callcontext,/* in */
        ORM_AppNodeDef          parent,     /* in */
        ORM_String              childname,  /* in */
        ORM_AppNodeDef          *child,     /* out */
        ORM_NodeTypeDef         *child_type /* out */
```

Fields:

parent	Opaque pointer to the virtual parent node.
childname	Name of the child (attribute), i.e. every printable char except '/'
child	Pointer where to store the reference to the node found
child_type	Pointer to type of node found.
returns	ORM_ENoError if child was found, else ORM_ENoSuchNode.

3.1.8 Function Type ORM_NodeTypeGetFunc

returns the type (enum ORM_NodeTypeDef) of the given node.

Declaration:

```
typedef ORM_NodeTypeDef (*ORM_NodeTypeGetFunc ) (
    ORM_AppNodeDef      node        /* in */
    );
```

Fields:

node	a pointer to a virtual node
returns	a valid type or ORM_NodeTypeUnknown.

3.1.9 Function Type ORM_NodeNameGetFunc

returns the name (ORM_String) of the given node.

Declaration:

```
typedef ORM_String (*ORM_NodeNameGetFunc ) (
    ORM_AppNodeDef        node        /* in */
    );
```

Fields:

node	a pointer to a virtual node
returns	a valid null terminated string of characters or NULL

3.1.10 Function Type ORM_NodeNotFoundTrapFunc

This function is kind of special by providing the application layer a chance, if the lookup of a node failed,to create that node.

Normally, referencing a non-existent node in the pathname of an ORM request is treated as an error, except this is an internal ORM restore request. Reloading an ORM tree into an application may encounter subtrees, which where dynamically created by the application during a previous run (usually via a *New* subtree).

This function is totally application dependant and is not covered by the ORM-SSL other than via this hook.

Declaration:

```
typedef ORM_Status (*ORM_NodeNotFoundTrapFunc) (
    ORM_NodeDef           parent,     /* in */
    ORM_String            name,       /* in */
    ORM_RequestTypeDef    request,    /* in */
    ORM_AppNodeDef        *newnode    /* out */
    );
```

Fields:

parent	Reference to parent node
name	Name of node not found under this parent.
request	Kind of ORM request (get/set/dump/restore) causing this lookup failure.
newnode	Where to store the reference to the new node, if one was created.
returns	ORM_ENoError, if a node with the given name was created else ORM_ENoSuchNode.

3.1.11 Structure ORM_NodeFuncDef

This structure bundles the virtual tree related functions for passing to ContextInitialize

Note: The ORM_NodeNotFoundTrapFunc is not included in this function array, because it is application special anyway and must be passed explicitly, see ContextInitialise()

Declaration:

```
typedef struct ORM_NodeFuncTag {
    ORM_NodeLookUpFunc          lookup;
    ORM_NodeChildNextFunc       childnext;
    ORM_NodeChildByNameFunc     childbyname;
    ORM_NodeTypeGetFunc         typeget;
    ORM_NodeNameGetFunc         nameget;
} ORM_NodeFuncDef;
```

3.1.12 Application Handles

The following two function types are used to link the virtual nodes in the tree to (parts of) actual application data instances, visible to the ORM support layer as opaque handles. When an application handle is requested from the application layer, *real things* happen to start and it is assumed, that the instances are valid and available, until explicitly released by the ORM layer. The handles together with the aspect (identifying the type of handle to the application) will be passed to the application specific functions, when actual values have to be accessed (either for *get* or *set*). If these functions are not set in the ORM context, NULL will be passed into those calls for both, the handle and the handleclass.

3.1.13 Function Type ORM_HandleGetFunc

Request (and lock) an actual handle (pointer to an application level instance) and a handleclass based on the current virtual node and the current principal.

Declaration:

```
typedef void status (*ORM_HandleGetFunc) (
    ORM_AppCallContextDef   callcontext,/* in */
    ORM_AppNodeDef          node,       /* in */
    ORM_RequestTypeDef      op,         /* in */
    ORM_AppHandleDef        *handle,    /* out */
    ORM_AppAspectDef        *aspect     /* out */
    );
```

Fields:

callcontext	is an opaque pointer to the application specific call context provided with the Do_Request function.
node	Pointer to current Node.
op	Operation Code, e.g. ORM_Request
handle	Pointer, where to store the handle reference

aspect	Pointer, where to store the aspect reference
returns	ORM_ENoError if no error occured or any of the ORM error codes.

3.1.14 Function Type ORM_HandleReleaseFunc

Returns a given handle back to the application layer. This should be more understood as an *unlock* operation than a free!

Declaration:

```
typedef void (*ORM_HandleReleaseFunc ) (
    ORM_AppCallContextDef   callcontext,/* in */
    ORM_AppHandleDef        handle,     /* in */
    ORM_AppAspectDef        aspect,     /* in */
    ORM_ReleaseTypeDef      op          /* in */
    );
```

Fields.

callcontext	is an opaque pointer to the application specific call context provided with the Do_Request function.
handle	a handle obtained via a call to HandleGet
aspect	Aspect as returned from HandleGet
op	Operation Code, e.g. ORM_Request

3.1.15 Function Type ORM_ObjectLinkGetFunc

Retrieve the Object Link from a node of type Object given the node, the handle and the aspect. The standard Handle Layer functions just return the link stored in the corresponding field in the node struct.

Declaration:

```
typedef ORM_Status (*ORM_HandleObjectLinkGetFunc ) (
    ORM_AppNodeDef          node,       /* in */
    ORM_AppHandleDef        handle,     /* in */
    ORM_AppAspectDef        aspect,     /* in */
    ORM_AppLinkDef *        link        /* out */
    );
```

Fields:

node	Reference to node of Object Type
handle	Reference to application defined handle as returned from HandleGet
aspect.	Reference to application defined aspect as returned from Han-

link	Location where to store the reference to the stringified link information
returns	ORM_ENoError if successfull, else ORM_InvalidOperation, if the node is not of type Object

3.1.16 Function Type ORM_AttributeDescrGetFunc

Retrieve the opaque reference unique to a node of type Attribute (usually the attribute descriptor), given the node, the handle and the aspect. The standard Handle Layer functions just return the pointer stored in the corresponding field in the node struct.

Declaration:

```
typedef ORM_Status (*ORM_HandleAttributeDescrGetFunc ) (
    ORM_AddressedRef          node,       /* in */
    ORM_AppHandleRef          handle,     /* in */
    ORM_AppAspectRef          aspect,     /* in */
    ORM_AppAttributesDescrRef *attribdesc /* out */
    );
```

Fields:

node	Reference to node of Object Type
handle	Reference to application defined handle as returned from HandleGet
aspect	Reference to application defined aspect as returned from HandleGet
attribdesc	Location where to store the reference to the attribute information
returns	ORM_ENoError if successfull, else ORM_InvalidOperation, if the node is not of type Object

3.1.17 Structure ORM_HandleFuncDef

This structure bundles the handle related functions for passing to ContextInitialize

Declaration:

```
typedef struct ORM_HandleFuncTag {
    ORM_HandleGetFunc                get;
    ORM_HandleReleaseFunc            release;
    ORM_HandleObjectLinkGetFunc      link;
    ORM_HandleAttributeDescrGetFunc  attrib;
    } ORM_HandleFuncDef;
```

3.1.18 Accessing Application Data: Aspects

the following group of functions (function types) has to be provided to access actual values of the application either for retrieval or for updating. All functions in this group are mandatory, if the ORM protocol layer is used.

3.1.19 Function Type ORM_AspectCallGetFunc

This function retrieves an *aspect* from the application layer, e.g. a reference to a blob of native application data (a pointer to a (part of) an application data structure, or a response buffer ....). The ORM protocol layer calls this function once for every unique handle/aspect combination (and not per Attribute) within a single AttributeGet Request. If the HandleGet Function returns a different pair or there are no more attribute nodes to process, the current aspect is released!

Declaration:

```
typedef ORM_Status (* ORM_AspectCallGetFunc) (
    ORM_AppHandleDef     handle,     /* in */
    ORM_AppAspectDef     aspect,     /* in */
    ORM_AppDataPtrDef    *current    /* out */
    );
```

Fields:

handle	Handle as retrieved from HandleGet
aspect	Aspect Reference, as retrieved from HandleGet
current	Where to store the reference to the current value (opaque)

3.1.20 Function Type ORM_AspectCallInitFunc

This function requests an *aspect* container from the application layer, e.g. a reference to a blob, where new attribute values can be selectivly written to to perform AttributeSet requests. In addition hte application layer may return a reference to the current aspects values (cmp CallGet), which is passed unchanged to the CallSet routine. The ORM protocol layer calls this function once for every unique handle/aspect combination (and not per Attribute) within a single AttributeSet Request. If the HandleGet Function returns a different pair for a node or there are no more attribute nodes to process, the CallSet function is called (Note: AspectRelease is only called for aspects retrieved via CallGet!) The

ORM SSL Implementation of these functions copies the current values and returns a reference to this copy in *new* and a reference to the current values in *current*.

Declaration:

```
typedef ORM_Status (* ORM_AspectCallInitFunc) (
    ORM_AppHandleDef     handle,      /* in */
    ORM_AppAspectDef     aspect,      /* in */
    ORM_AppDataPtrDef    *new,        /* out */
    ORM_AppDataPtrDef    *current     /* out */
    );
```

Fields:

handle	Handle as retrieved from HandleGet
aspect	Aspect Reference, as retrieved from HandleGet
new	Where to store the reference to the native blob to update with new attribute values (opaque)
current	Where to store the reference to the current aspect (opaque)

3.1.21 Function Type ORM_AspectCallSetFunc

This function is called to actually apply the new attribute values for the current aspect by the application layer. It is up to the aspect/application layer, to check the values in the request structure for validity and consistency and to determine which attributes got new values (by comparison with the *current* values). In addition it is the responsibility of the aspect/application layer to deallocate any structures allocated by AspectCallInit. Only if the Set-Function is not called, the call to AspectRelease is performed.

The ORM protocol layer calls Set-function once for every unique handle/aspect combination (and not per Attribute) within a single AttributeSet Request. If the HandleGet Function returns a different pair for a node or there are no more attribute nodes to process, the CallSet function is called (Note: AspectRelease is only called for aspects retrieved via CallGet!) The ORM SSL Implementation of these functions copies the current values and returns a reference to this copy in *new* and a reference to the current values in *current*.

Declaration:

```
typedef ORM_Status (* ORM_AspectCallSetFunc) (
    ORM_AppHandleDef     handle,      /* in */
    ORM_AppAspectDef     aspect,      /* in */
    ORM_AppDataPtrDef    new,         /* in */
    ORM_AppDataPtrDef    current,     /* in */
    ORM_String           *reDetail    /* out */
    );
```

aspect	Aspect Reference, as retrieved from HandleGet
request	Where to store the reference to the native blob to update with new attribute values (opaque)
current	Where to store the reference to the current aspect (opaque)
rsdetail	Where to store a textual hint, why the call failed, if any.
returns	ORM_ENoError if new values could be applied successfully, else ORM_ERange.

3.1.22 Function Type ORM_AspectReleaseFunc

Used to tell the application layer, that the reference retrieved via an AspectGet or AspectInit call is no longer needed anymore by the ORM layer. This function is called, when GetHandle returns a new handle aspect call within a AttributeGet processing or a conversion in an AttributeSet processing failed.

Declaration:

```
typedef void (* ORM_AspectReleaseFunc) (
    ORM_HandleGet       handle,     /* in */
    ORM_AspectGet       aspect,     /* in */
    ORM_AspectRefBlob   current,    /* in */
    ORM_RequestTypeDef  reqtype     /* in */
    );
```

Fields:

handle	Handle as retrieved from HandleGet
aspect	Aspect Reference, as retrieved from HandleGet
current	Reference to data as returned from AspectCallInit or AspectCallGet.
reqtype	ORM_RequestGet or ORM_RequestSet depending whether this dataptr resulted from an AspectGet or AspectInit call.

3.1.23 ORM_AspectFuncDef

This function groups the function pointers of the aspect layer

Declaration:

```
typedef struct ORM_AspectFuncTag {
    ORM_AspectCallGetFunc    callget;
    ORM_AspectCallInitFunc   callinit;
    ORM_AspectCallSetFunc    callset;
    ORM_AspectReleaseFunc    release;
} ORM_AspectFuncDef;
```

3.1.24 Attribute Functions

The following group of functions is called to actually perform the the single attribute Get/Set and the corresponding conversions between the applications native and the ORM (ascii) presentation.

3.1.25 Data Structure: ORM_AttributeInfoDef

This structure is used to return the all the meta information and the actual value of an attribute. It is passed by reference to the application/attribute layer to be filled. Note: The string pointers do not point to valid buffers, when passed to the attribute layer!

```
<name>   Name of the attribute
<type>   INT[1,2,4,8], REAL[32/64], STRING, HEXOCT,
         OPAQUE, NONE
<value>  the current value in its ascii presentation
<range>  Optional: The range string
<unit>   Optional: The unit string
```

Declaration:

```
typedef struct ORM_AttributeInfoTag {
    ORM_String          value;
    ORM_String          name;
    ORM_String          field;
    ORM_String          range;
    ORM_String          unit;
} ORM_AttributeInfoDef;
```

3.1.26 Function Type ORM_AttributeNativeToStringFunc

This function converts the applications native value of an *attribute*, specified by *handle*, *aspect* and the attribute descriptor to a C-string (ORM_String).

Declaration:

```
typedef CRM_Status (*CRM_AttributeNativeToStringFunc)(
        CRM_AppHandleDef        handle,         /* in */
        CRM_AppAspectDef        aspect,         /* in */
        CRM_AppAttribDescrDef   attribdescr,    /* in */
        CRM_AppDataPtrDef       dataptr,        /* in */
        CRM_String              *strvalue       /* out */
        );
```

Fields:

handle	Handle as obtained from the last call to HandleGet or NULL.
aspect	Aspect as returned from the last call to HandleGet or NULL
attribdescr	*Attribute Descriptor as returned form AttribDescrGet* call.
dataptr	Opaque Pointer as returned from AspectGetCall.
strvalue	Where to store the reference to the converted value.
returns	ORM_ENoError (Null) if conversion was successfull, else a valid ORM Error return code.

3.1.27 Function Type ORM_AttributeNativeToInfo

This function performs the same as the previous function *ORM_AttributeNativeToString, except that it also provides* the additional meta information to this attribute, as far as available.

Declaration:

```
typedef CRM_Status (*CRM_AttributeNativeToInfoFunc)(
        CRM_AppHandleDef        handle,     /* in */
        CRM_AppAspectDef        aspect,     /* in */
        CRM_AppAttribDescrDef   attribdescr,/* in */
        CRM_AppDataPtrDef       dataptr,    /* in */
        CRM_AttributeInfoDef    info        /* in, indirect out */
        );
```

Fields:

handle	Handle as obtained from the last call to HandleGet or NULL.
aspect	Aspect as returned from the last call to HandleGet or NULL
attribdescr	*Attribute Descriptor as returned form AttribDescrGet* call.
dataptr	Opaque Pointer as returned from AspectGetCall.
extref	Pointer to structure, where to store the string references.

returns ORM_ENoError (Null) if conversion was successfull, else a valid ORM Error return code.

3.1.28 Function Type ORM_AttributeStringToNativeFunc

This function converts an ORM_String value for an attribute into the applications native presentation. The conversion should be done into the structure (dataptr) obtained by a call to AspectCallInit().

Declaration:

```
typedef ORM_Status (*ORM_AttributeStringToNativeFunc) (
        ORM_AppHandleDef        handle,     /* in */
        ORM_AppAspectDef        aspect,     /* in */
        ORM_AppAttribDescrDef   attribdescr,/* in */
        ORM_AppDataPtrDef       dataptr,    /* in, indirect out */
        ORM_String              strvalue    /* in */
        );
```

Fields:

handle	Handle as obtained from the last call to HandleGet or NULL.
aspect	Aspect as returned from the last call to HandleGet or NULL
attribdescr	*Attribute Descriptor as returned form AttribDescrGet* call.
dataptr	Opaque Pointer as returned from AspectGetCall.
strvalue	New value as a C-String (ascii).
returns	ORM_ENoError (Null) if conversion was successfull, else a valid ORM Error return code.

3.1.29 Structure ORM_AttributeFuncDef

This structure bundles the attribute related functions for passing to ContextInitialize

Declaration:

```
typedef struct ORM_AttributeFuncTag {
    ORM_AttributeStringToNativeFunc     stringtonative;
    ORM_AttributeNativeToStringFunc     nativetostring;
    ORM_AttributeNativeToInfoFunc    infotostring;
    } ORM_AttributeFuncDef;
```

3.1.30 Structure ORM_ContextDef

This is an internal structure to ORM and opaque to the application layer. It stores the function pointers and the information of the root node.

Note: This structure and the related procedure definitions may change

authfuncs	Pointer to list of authentication related functions or NULL, if no application specific authentication is needed.
notfound	No description

3.1.32 ORM_ContextRelease

Release an Application Context.

Prototype:

```
void
ORM_ContextRelease( ORM_ContextDef context);
```

Parameters:

context	Pointer to application context as obtained from ORM_ContextInitialize

3.1.33 ORM_DoRequest

This function calls the protocol layer to parse an ORM request received and act on it accordingly via upcalls to functions in the application context, i.e. this is the function to be dispatched, when ORM requests are received on a server port.

Prototype:

```
ORM_Status
ORM_DoRequest(
    ORM_ContextDef      appctxt,
    void                *callctxt,
    ORM_RequestBuffer   request,
    long                reqlen,
    ORM_ResponseDef     response,
    long                *maxresp
    );
```

Parameters:

appctxt	The application context reference as returned from ORM_ContextInitialize.
callctxt	An arbitrary call context (reference) maintained by the application layer and passed to the authentication, node and handle upcalls.
request	Pointer to received ORM request
reqlen	Length of request buffer in bytes
response	Pointer to allocated response buffer

maxresp	Reference to maximum response buffer length in bytes, on return, points to number of bytes used in response buffer

3.2 ORM Node Layer

This section was generated from *<stdin>* by CDOC on Fri Jan 27 19:59:34 1995.

The ORM Node layer adds another level of ORM application/server support, as it actually maintains a tree structure to access the application level datastructures.

This level is accessed from the application/server level via the *ORM_Node...* functions to actually build/destroy the tree of objects, components and attributes.

On the other side it is called from the protocol level and frees up the application to provide the appropriate functions for navigation and name space/entity management itself.

3.2.1 Application Handles

The nodes of the node layer provide a tree structured view to application/server level data, but they (usually) do not contain the actual data. A link to the actual instances of application level data is maintained by *handles* and *aspects*. Both are opaque to the ORM-Node level but are interpreted at the layer on top of ORM-Node. Typically the handle is a pointer to some application level instance, and the aspect is a pointer, index or type identifier, which identifies the type of the instance

3.2.2 The ORM_Node Structure

Instances of this structure maintain the tree of virtual components, objects and attributes

Every node has a name and a type, identifying the three different entity types: Object, Component or Attribute. Object and Attribute nodes are leaf nodes, e.g. they can't have children.

In addition, every node has a parent and a next pointer, to link the actual tree structure. Only component nodes have a pointer to the list of children.

Object Nodes have an additional attribute, called the Link (or Link-Info which usually is a stringified NOR).

Attribute Nodes reference a single attribute by, which is characterize by additional information like

- a value type, which describes the kind of value e.g. integer (different sizes), real (sizes!), string, *single-selection* or *multiple choice*
- a value mode, specifying this attribute as read-only read-write, write-only or persistant.
- *hints* section, which contains additional information for use by the user-interface creator, e.g. valid ranges for this value and a unit string. Both values are optional.

The nodes provide a tree structured view to application/ server level data, but they (usually) do not contain the actual data.

3.2.3 Struct NodeDef

Declaration:

```
typedef struct ORM_NodeTag {
    ORM_NodeTypeDef      type;
    short                flag;
    char                 *name;
    struct ORM_NodeTag   *parent;
    struct ORM_NodeTag   *next;
    ORM_AppHandleDef      handle;
    ORM_AppAspectDef      aspect;
    union {
        struct
            struct ORM_NodeTag *first;
            struct ORM_NodeTag *last;
            } comp;
        struct
            void     *descr;
            } attrib;
        struct {
            char     *link;
            } object;
        } u;
} *ORM_NodeDef;
```

Fields:

type	indentifies the type of entity, this node describes, i.e. ORM_NodeType{Object, Component Attribute, Unknown}
flag	Internal use
name	The name of the node (object, component or attribute name
parent	pointer to the parent in the tree, NUL for the root of the tree.
next	pointer to next sibbling in chain. This defines the order in which nodes of a given type appear in the response
handle	an opaque pointer for use by the upper layers
aspect	another opaque identifier for use by the upper layers
u.comp	union variant for component nodes
u.comp.first	pointer to first child of this component node
u.comp.last	pointer to last child of this component node
u.attrib	union variant for attribute nodes
u.attrib.descr	opaque pointer for use by upper layers

u.object.link	pointer to stringified link-address of this object (NOR), e.g. the *hyperlink*

3.2.4 ORM_NodeCreate

Creates a new unlinked node. Usually only used by convenience functions and to create the root node.

Prototype:

```
ORM_NodeDef
ORM_NodeCreate( ORM_String       name,    /* in */
                ORM_NodeTypeDef type     /* in */
              );
```

Parameters:

name	The name of this node (for navigation)
type	The type of this node. This type also determines which functions can be applied to this node later on.

3.2.5 ORM_NodeDelete

Deletes the given node and all its children e.g. returns the space allocated Note: if the nodes parent pointer is not NULL, the node will not be deleted.

Prototype:

```
int
ORM_NodeDelete( ORM_NodeDef node     /* in */
              );
```

Parameters:

node	The node (and the subtree) to delete

3.2.6 ORM_NodeAttach

Attaches a node (and its subtree) into an existing tree as a new subtree. Every node (subtree) is in at most 1 tree!

Prototype:

```
int
ORM_NodeAttach( ORM_RelationDef relation,       /* in */
                ORM_NodeDef     relative,       /* in */
                ORM_NodeDef     subtree         /* in */
              );
```

Parameters:

relation : Flag either ORM_NodeSibbling or ORM_NodeChild, specifying the role of the *relative* node, e.g. its a sibbling or its the parent of the subtree to attach. If its a parent, the new node will be attached at the end of all children, if its a sibbling, it will be placed right before this child.

relative : an existing node, either parent of sibbling

subtree No description

3.2.7 ORM_NodeDetach

Detaches a subtree from the current root tree. This always has to be called, before a subtree is actually deallocated. The subtree may also be reattached in the same tree again after this call

Prototype:

```
int
ORM_NodeDetach(  ORM_NodeDef subtree      /* in */
                 );
```

No parameter descriptions are available.

3.2.8 ORM_NodeHandleSet

Sets the handle in the given node (see also ORM_Node<convenience functions>)

Prototype:

```
void
ORM_NodeHandleSet(  ORM_NodeDef          node,      /* in */
                    ORM_AppHandleDef     handle     /* in */
                    );
```

Parameters:

node Reference to node structure of any type.

handle Reference to opaque handle.

3.2.9 ORM_NodeHandleGet

Retrieves the handle from a given node

Prototype:

```
int
ORM_NodeHandleGet(  ORM_NodeDef          node,      /* in */
                    ORM_AppHandleDef     *handle    /* out */
                    );
```

No parameter descriptions are available.

3.2.10 ORM_NodeAspectSet

Sets the aspect in the given node (see also ORM_Node<convenience functions>)

Prototype:

```
void
ORM_NodeAspectSet(  ORM_NodeDef          node,        /* in */
                    ORM_AppAspectDef     aspect       /* in */
                    );
```

Parameters:

node	Reference to node structure of any valid node type.
aspect	Reference to opaque aspect description.

3.2.11 ORM_NodeAspectGet

Retrieves the aspect from a given node

Prototype:

```
int
ORM_[illegible](  ORM_NodeDef          node,                /* in */
                  ORM_AppAspectDef     *aspect              /* out */
                  );
```

No parameter descriptions are available.

3.2.12 ORM_NodeAttributeDescrSet

Sets the attribute description of an attribute node

Prototype:

```
int
ORM_[illegible](  ORM_NodeDef              node,        /* in */
                  ORM_AppAttribDescrDef    attrib       /* in */
                  );
```

Parameters:

node	Reference to node structure of type Attribute.
attrib	Reference to opaque attribute description

3.2.13 ORM_NodeAttributeDescrGet

Gets the attribute description of an attribute node

Prototype:

```
int
ORM_NodeAttributeDescrGet( ORM_NodeDef              node,       /* in */
                           ORM_AppAttribDescrDef    *attrib     /* out */
                           );
```

No parameter descriptions are available.

3.2.14 ORM_NodeObjectLinkSet

Sets the link of an object node

Prototype:

```
int
ORM_NodeObjectLinkSet( ORM_NodeDef node,          /* in */
                       ORM_String  link           /* in */
                       );
```

Parameters:

node	Reference to node structure of type Object.
link	Stringified version of the address/nor to call this object.

3.2.15 ORM_NodeObjectLinkGet

Gets the linkaddress of an object node

Prototype:

```
int
ORM_NodeObjectLinkGet( ORM_NodeDef node,          /* in */
                       ORM_String  *link          /* out */
                       );
```

No parameter descriptions are available.

3.2.16 ORM_NodeObjectAdd

for an explanations of paramters, see above. Return created node if operation succeeded else NULL.

Prototype:

```
ORM_NodeDef
ORM_NodeObjectAdd(      ORM_RelationDef     relation,       /* in */
                        ORM_NodeDef         relative,       /* in */
                        ORM_String          name,           /* in */
                        ORM_AppHandleDef    handle,         /* in */
                        ORM_AppAspectDef    aspect,         /* in */
                        ORM_String          linkaddr        /* in */
                  );
```

No parameter descriptions are available.

3.2.17 ORM_NodeComponentAdd

Prototype:

```
ORM_NodeDef
ORM_NodeComponentAdd(   ORM_RelationDef     relation,       /* in */
                        ORM_NodeDef         relative,       /* in */
                        ORM_String          name,           /* in */
                        ORM_AppHandleDef    handle,         /* in */
                        ORM_AppAspectDef    aspect          /* in */
                        );
```

No parameter descriptions are available.

3.2.18 ORM_NodeAttributeAdd

Prototype:

```
ORM_NodeDef
ORM_NodeAttributeAdd(   ORM_RelationDef     relation,       /* in */
                        ORM_NodeDef         relative,       /* in */
                        ORM_String          name,           /* in */
                        ORM_AppHandleDef    handle,         /* in */
                        ORM_AppAspectDef    aspect,         /* in */
                        ORM_AppAttribDescrDef attribdescr   /* in */
                        );
```

No parameter descriptions are available.

3.3 ORM Aspect Layer

This section was generated from *<stdin>* by CDOC on Sun Jan 29 17:00:51 1995.

The ORM aspect layer adds another level of ORM application/server support on top of the ORM Node/Handle layer, and supports the retrieval and modification of aspects, i.e. groups of attributes from or into application data structures, once those have been registered with this layer.

This level has no additional (down-call) functions but defines data structures to be provided by the application layer. These are then accessed/used by the aspect upcall functions, if those have been registered with the ORM protocol layer.

The Aspect layer implementation of the ORM-SSL works as follows:

On AspectCallGet requests, just a pointer is returned which points at offset bytes (as set in the aspect descriptor) from the beginning of the handle. On AspectCallInit calls, a copy of the aspect, e.g. size bytes from the area pointed to by handle, starting from offset, is taken into a private memory area. This copy is then passed to the Attribute conversion routines to write the new values into. On AspectCallSet calls, the application level set function as denoted by the aspect descriptor is called and the private copy (request structure) is released afterward.

3.3.1 Function Type ORM_AspectSetFunc

This function is called from the aspect layer to actually apply the new attribute values to the application layer and/or initiate the requested state changes. This function usually should not block, e.g. should not wait until the initiated state change is completed. Any kind of intermediate state should instead be visible to a client on request (i.e. not STOPPED -> STARTED, but STOPPED -> STARTING -> STARTED, if starting implies a *heavier* operation.

Declaration:

```
ORM_Status
typedef (*ORM_AspectSetFunc)(
    ORM_AppHandleDef        handle,         /* in */
    ORM_AttributeDescrDef   aspect,         /* in */
    ORM_AppDataPtrDef       request,        /* in */
    ORM_AppDataPtrDef       current,        /* in */
    ORM_String              *errortext      /* out */
    );
```

Fields:

handle	the handle as returned from HandleGet
aspect	Reference to the aspectdescr.
request	Copy of the aspect as described by the aspectdescr updated with new values.

current	Reference to aspect within handle
errortext	Where to store a pointer to a short textual description if the requested values could NOT be applied.
returns	ORM_ENoError if all new values could be applied, or ORM_EParameterList if paremter set is inconsistent or ORM_EMissingAttribute if a mandatory attribute is NULL.

3.3.2 The ORM_AspectDescrDef

This descriptor maintains information about the application data structure (usually references by the ORM_AppHandle) or parts of it. It describes the binary size, the offset within the handle, and contains pointers to functions to actually retrieve or modify this aspect of the application instance.

Note: It is currently open, whether there should be a procedural interface to set up the aspect descriptor instead of providing a structure type definition to be passed initialized by the application code.

Declaration:

```
typedef struct _ORM_AspectDescrTag {
        char                    *name;
        size_t                  offset;
        size_t                  size;
        long                    flag;
        ORM_AspectSetFunc       setf;
        long                    appid;
        void                    *appext;
} ORM_AspectDescrDef;
```

Fields:

name	Pointer to name string, for identification mainly.
offset	The offset in bytes within the instance, where this aspect starts. This usually is the offset of a sub structure in the instance.
size	The size in bytes of the instance, the application handle pointer points to. For set-requests, the container for the new value is created by copying the handle, and inserting the new values in it.
flag	If set to ORM_AspectGetIndirect, the offset indicates the offset to a pointer, pointing to another structure of the above size.
setf	Pointer to function, which is called to apply (a set of) new values to an application instance.
appext	any value of pointer size the application wants to store with the aspect. This may be used to store a create_aspect function pointer.
appid	Opaque identifier, which may be used by the applications layer

3.4 ORM Attribute Layer

This section was generated from *<stdin>* by CDOC on Fri Jan 27 19:59:34 1995.

The ORM Attribute layer adds another level of ORM application/server support on top of the ORM node layer, by providing (list of) attribute descriptors, which simply initialized by the application code, allowe automatic conversion and generation of the attribute meta information, requested by the ORM protocol layer.

The implementation of the attribute layer in the ORM SSL assumes, that it is converting to and from a binary blob of data, identified by the (lower level) aspect descriptor. The goal of this layer is to reduce the coding effort needed by the application writer at this layer, just to provide some initialized descriptors and pass them to the ORM SSL via single calls per every instance created.

3.4.1 The ORM_AttributeDescriptorDef

This data structure describes a single attribute, e.g. its native type and mode, its size, pointers to conversion functions. In addition it maintains hooks for preset meta-info like - *Unit* and *Range*.

Declaration:

```
typedef struct ORM_AttributeDescrTag {
    ORM_String                          name;
    ORM_AttribTypeDef                   datatype;
    ORM_AttribModeDef                   accessmode;
    ORM_String                          range;
    ORM_String                          unit;
    size_t                              offset;
    size_t                              size;
    ORM_ConverterNativeToStringFunc     nativetostring;
    ORM_ConverterStringToNativeFunc     stringtonative;
    ORM_AppConverterArgDef              convarg;
} ORM_AttributeDescrDef;
```

Fields:

name	The name of the attribute.
datatype	The type of data of this attribute (ORM_AttributeTypeDef). This is a superset of the data types, the ORM protocol defines and used to determine implicit conversion routines.
mode	The allowed access modes of this attribute out of ORM_AttribMode values, e.g. read-only, write- only, read-write.
range	A string describing the allowed ranges for new values for read-write or write-only attributes only. This is a *ORM hint*, and as such optional
unit	A unit string (usually *ms*, *Mb*, etc.) which may be used by object specific user interface generators in any way, and by default if

present is placed behind the attribute value. This is also an *ORM hint* and as such optional.

conversion function A function pointer to an application specific conversion function, to convert between native and ORM presentations. *Note:* This is not to be confused with the similar functions of the ORM_Context structure. For the <conversion-function> to be called, the ORM_Node conversions functions have to be setup in the ORM_Context.

conversion-arg An opaque pointer to any argument, the conversion function may need to convert this attribute.

3.4.2 ORM_AttributeCreate

This function combines several actions required to register an attribute of a (new) instance with the ORM SSL, i.e. it creates an attribute node under the given parent (which must be of ORM_NodeTypeComponent) and attaches the attribute description and the handle information to it.

Prototype:

```
int
ORM_AttributeCreate(
                ORM_NodeDef                     relative,    /* in */
                ORM_RelationDef                 relation,    /* in */
                ORM_AttributeDescrListDef       attribdescr,/* in */
                ORM_AspectDescrListDef          aspectdescr,/* in */
                ORM_AppHandleDef                handle,      /* in */
                ORM_NodeDef                     *new         /* out */
                );
```

Parameters:

relative pointer to relative node. If *relation* is set ORM_NodeIsParent, then this has to be a node of ORM_NodeTypeComponent. If *relation* is set to ORM_NodeIsSibbling, then this node can be of any valid node type.

relation Either ORM_NodeIsParent, if the node *relative* should be the parent of the new attribute node, or ORM_NodeIsSibbling, if the new attribute node should be inserted after the *relative* node as a sibbling.

attribdescr No description

aspectdescr No description

handle Pointer to the application instance this attribute belongs to or ORM_HandleInherit (-1), if the handle should be taken from the parent (or its parent and so on).

new Pointer to new attribute node or NULL on failure.

3.4.3 ORM_AttributeDestroy

This function detaches the attribute node from the tree of nodes if any, deletes the node structure and deletes any depending structures, i.e. the attribute descriptor.

In the current implementation this function maps directly to ORM_NodeDestroy, but nevertheless this function should be called for attribute nodes created with functions of this layer to be able to deallocate any dynamic memory.

Prototype:

```
int
ORM_AttributeDestroy(    ORM_NodeDef attrnode);
```

Parameters:

attrnode	Pointer to attrbute node.

3.4.4 ORM_AttributeListCreate

This is another convenience functions to add a list of attributes to a component. The given node must be a of component type and is used as the parent for the new list of attributes (which is appended to the end of the list of child-nodes). The pointer to the attribute descriptor now points to an array of those descriptors, where the end of the array is marked by a descriptor whose name pointer is NULL.

Prototype:

```
int
ORM_AttributeListCreate(
        ORM_NodeDef                 parent,        /* in */
        ORM_AppHandleDef            handle,        /* in */
        ORM_AspectDescrDef          aspectdescr,   /* in */
        ORM_AttributeDescrListDef   attrdescrlist, /* in */
        long                        attrcount      /* in */
        );
```

Parameters:

parent	Pointer to an existing component node, who is the parent node of all newly created attribute nodes.
handle	Pointer to the application instance, all attribute belongs to or ORM_HandleInherit (-1), which indicates, that the actual handle is determined by the parent (which again may have its handle set to ORM_HandleInherit!)
aspectdescr	No description
attrdescrlist	Pointer to an array of ORM_AttributeDescr, with name=NULL in the last element if *attrcount* is < 0.

attrcount	The number of attribute descriptors in the list or the number of initial attributes from this list to attach to this node or -1, if the end of the list (array) should be determined by a NULL nodeinfo pointer.

3.5 ORM Attribute Conversion Support

This section was generated from *<stdin>* by CDOC on Sun Jan 29 18:13:38 1995.

This part or the ORM Server Support Layer provides functions for converting generic ORM data types between their native (binary) and the ORM (ASCII) presentation. The interface between the attribute and the conversion layer is defined by to function types, one for converting application native data into an ORM representation, one to convert ORM attribute value strings into the applications native presentation. Beside the conversion functions provided by the ORM-SSL, every application may provide its own special converters as long as their interfaces conform these function types.

3.5.1 Function Type ORM_ConverterNativeToString

This function is called to convert a single native value into its string representation. In addition to the value string it may generate the range and unit strings, if the pointer values passed are non-null. If the converter function returns NULL in these pointers, the lower (attribute) layer may provide default strings if any.

Memory Allocation: The memory to hold the converted string value(s) has to be provided by the converter function. It is reasonable to use static memory for this purpose, because before the converter function is called again, the ORM protocol layer will copy the strings returned.

Declaration:

```
typedef ORM_Status (* ORM_ConverterNativeToStringFunc)(
            ORM_AppDataPtrDef          ptr,          /* in */
            size_t                     size,         /* in */
            ORM_AttributeDescrListDef  datatype,     /* in */
            ORM_AppConverterArgDef     convarg,      /* in */
            ORM_String                 *strvalue,    /* out */
            ORM_String                 *strrange,    /* out */
            ORM_String                 *strunit      /* out */
            );
```

Fields:

ptr	Address of native data element (e.g. attribute value)
size	Byte-size of data element
datatype	One of the ORM_AttributeTypes identifying the type of the native data element and its mapping to an ORM Protocol data type (??is this overloaded ??)
convarg	Any kind of argument (pointer) for this converter (as provided with the attribute descriptor for ex.)
strvalue	Where to store the pointer to the converted value string.
strrange	Where to store the reference to the optional range string.

strunit Where to store the reference to the optional unit string.

3.5.2 Function Type ORM_ConverterStringToNative

This function is called to convert a single ORM string value into its native presentation. The pointer for the result usually points into a set of different attributes, e.g. an aspect, which usually is a (partial) copy of some application data instance.

Memory Allocation: The destination pointer provided references some valid memory (e.g. an aspect), but for references (the native value is a C-string for ex.), there is usually not enough space for the referenced value. This space must be allocated/provided by the converter itself. It is legal, to reference the original string as passed in to the converter function, but then the AspectCallSet function should make a copy, if the string is needed beyond this call.

Declaration:

```
typedef ORM_Status (* ORM_ConverterStringToNativeFunc)(
            ORM_AppDataPtrDef       dest,          /* in */
            size_t                  size,          /* in */
            ORM_AttributeTypeDef    datatype,      /* in */
            ORM_AppConverterArgDef  converg,       /* in */
            ORM_String              strvalue       /* in */
            );
```

Fields:

dest	Address/destination of native data element (e.g. attribute value)
maxsize	Maximum byte-size of data element
datatype	One of the ORM_AttributeTypes identifying the type of the native data element
converg	Any kind of data (pointer) for this converter as provided with the attribute descriptor
strvalue	The new attribute value in its ascii presentation.
returns	ORM_ENoError if conversion was successfull and the resulting attribute value is valid or ORM_ERangeError.

3.5.3 ORM Built In Conversion Functions

The following functions are provided to convert generic C datatypes between their ORM and their native presentation. In addition sub functions are provided to support the special ORM SELECT and MCHOICE types, which are called by the generic converters. Along with these sets two new data structure (types) are introduced.

3.5.4 Function ORM_GenericNativeToString

This function converts standard C-data types into their ASCII presentation. It returns only the converted value, but does not support the range and unit parts (e.g. returns NULL for those, if requested). In case of SELECT or MCHOICE functions, this function calls the related ORM_Select.. or ORM_MChoice functions.

Note: It is currently open, whether the conversion argument *convarg* may be used to specify a format string a la *printf*. Furthermore it is currently open, whether a NULL conversion function in the attribute descriptor should be directed to this (default) function.

Arguments as for ORM_ConverterNativeToString!

Prototype:

```
ORM_Status ORM_GenericNativeToString(
        ORM_AppDataPtrDef          ptr,          /* in */
        size_t                     maxsize,      /* in */
        ORM_AttribTypeDef          type,         /* in */
        ORM_AppConverterArgDef     convarg,      /* in */
        ORM_String                 *strvalue,    /* out */
        ORM_String                 *rangevalue,  /* out */
        ORM_String                 *strunit      /* out */
        );
```

No parameter descriptions are available.

3.5.5 Function ORM_GenericStringToNative

This function converts ASCII C-strings into standard C-datatypes. In case of SELECT or MCHOICE functions, this function calls the related ORM_Select.. or ORM_MChoice functions.

Note: It is currently open, whether the conversion argument *convarg* may be used to specify a format string a la *sscanf*. Furthermore it is currently open, whether a NULL conversion function in the attribute descriptor should be directed to this (default) function.

Arguments as for ORM_ConverterStringToNative!

Prototype:

```
ORM_Status ORM_GenericStringToNative(
        ORM_AppDataPtrDef          ptr,          /* in */
        size_t                     maxsize,      /* in */
        ORM_AttribTypeDef          type,         /* in */
        ORM_AppConverterArgDef     convarg,      /* in */
        ORM_String                 strvalue      /* in */
        );
```

No parameter descriptions are available.

3.5.6 Structure ORM_StringMapDef

This type of structure is used to map strings to binary values and vice versa. It may be used to convert internal flags and states to *friendly* names. StringMaps must be terminated by an entry with *name* set to NULL.

Declaration:

```
typedef struct ORM_StringMapTag {
    ORM_String  name;
    ORM_Key     key;
} ORM_StringMapDef;
```

Fields:

name	Friendly name for this key.
key	The binary native value of the key

3.5.7 ORM_StringMapToString

This function maps a value key to a string using the given StringMap. It returns the string of that map entry, whose key is equal to the given key, else it returns the string passed in *notfound*.

Prototype:

```
ORM_String
ORM_StringMapToString( ORM_StringMapDef     map,
                       ORM_Key              key,
                       ORM_String           notfound);
```

Parameters:

map	Pointer to a sequence of map entries
key	Binary key value.
notfound	string to give back, if none of the keys in the map matched.

3.5.8 ORM_StringMapToKey

This function maps a string value to a binary key using the given StringMap. It returns the key of that map entry, whose string is equal to the given key, else it returns the key passed in *invalidkey*.

Prototype:

```
ORM_Key
ORM_StringMapToKey( ORM_StringMapDef     map,
                    ORM_String           name,
                    ORM_Key              invalidkey);
```

Parameters:

map	Pointer to a sequence of map entries
name	No description
invalidkey	No description

3.5.9 Structure ORM_StateMapDef

This structure is used to map *states* into strings, where a *state* is assumed to have a distinct set of possible next states, depending on the current value. E.g. this structure can be used to derive the set of possible new values i.e. it can provide the *range* value for a state attribute.

Otherwise it is used similar to the simpler StringMap structure. StateMaps must be terminated by an entry with *name* set to NULL.

Declaration:

```
typedef struct _ORM_StateMapTag { /* not of the U.S.A.!! */
    ORM_String  name;
    ORM_Key     state;
    ORM_String  validnexts;
} *ORM_StateMapDef;
```

Fields:

name	Friendly name for this key.
state	The binary native value of this state
validnexts	String of comma separated names of next valid states which may follow this state.

3.5.10 ORM_StateMapToString

Convert an encoding of a state into a *friendly* name using the given statemap. If the state could not be found, the string passed in *notfound* is returned.

Prototype:

```
ORM_Status
ORM_StateMapToString (  ORM_StateMapDef map,
                        ORM_Key         state,
                        ORM_String      notfound);
```

Parameters:

map	Pointer to a (name=NULL) terminated state map.
state	the binary state
notfound	string to return, if none of the entries in the map had exactly the given state key.

3.5.11 ORM_StateMapToKey

Convert a string representation of a state into a native encoding of a *state using the given statemap*. If the string could not be found, the state passed in *invalidstate* is returned.

Prototype:

```
ORM_Key
ORM_StateMapToKey ( ORM_StateMapDef map,
                    ORM_String      name,
                    ORM_Key         invalidstate);
```

Parameters:

map	Pointer to a (name=NULL) terminated state map.
name	No description
invalidstate	No description

3.5.12 ORM_StateMapNextByKey

Return the comma separated list of valid next states given the current state.

Prototype:

```
ORM_String
ORM_StateMapNextByKey(  ORM_StateMapDef map,
                        ORM_String      state);
```

Parameters:

map	Pointer to a (name=NULL) terminated state map.
state	the binary state

3.6 ORM Dump & Restore Support

This section was generated from *<stdin>* by CDOC on Fri Jan 27 19:59:34 1995.

This module of the ORM Server Support Library supports the dump and restore of complete subtrees, and therefore can be used to save the current configuration to a persistant storage media (i.e. the MSF Warehouse) and reload it from there. The actual IO functions are currently not supported by this layer or the support library at all!

Dump and Restore are functions of the ORM SSL and not of the ORM protocol (i.e. there is no *DUMP* or *RESTORE* request defined in the protocol).

This implies, that these functions have to be dispatched out of the application layer explicitly. One (intended) way to dispatch those functions interactively is to provide pseudo components in every subtree, which should be independent storable/reloadable. These contain the required parameters like Warehouse location or version name as attributes. An Attribute-Set request to this subtree then results in the execution of the corresponding function.

Under the layered view of the ORM SSL, these two functions belong to the protocol layer, as they use (nearly) the same functionality of the higher layers via upcalls.

3.6.1 General Model:

Starting from a given node, which is used as the root of the relevant subtree to dump, all components, object links and writable attributes with their meta information are recursivly extracted relative to the current subtree root. The extended/meta information on the persistent media can be used to interprete the stored attributes and apply changes to the stored version without the ORM server/application alive but through special clients (by an ORM/Warehouse gateway for example).

The dumped ORM tree can be used to reload the whole subtree at any time, by providing the node and call the *restore* function of the ORM SSL (which is a special kind of Set-Request).

This special kind of SET request creates a new situation, as components (or any new subtree) may have been created dynamically by the ORM server application on request. On the next cold start of the application, these subtrees do not exist.

This results in failed lookup requests by the ORM protocol layer, which usually is treated as an error (remember: ORM-P has no direct support for object/component creation, but this is emulated by sets of writeonly attributes in separate subtrees, i.e. *New..*). To handle this case, the application can provide a special function during application context setup to create new instances including the ORM subtree (ORM_NodeNotFoundTrapFunc()).

A parameter is passed to this creation function, which indicates, whether this situation was caused by a regular ORM protocol request or by an internally generated *restore* request, so the application code can still decide to refuse the creation.

3.6.2 ORM_Dump

This function extracts the ORM entities in the subtree pointed to by *subtree* into the character buffer, so it can be used by a later ORM_Restore function (or can be used as a subrequest in a regular ORM protocol request).

It is the responsibility of the caller to provide a sufficient buffer, which can hold the subtree information of the given depth!

Prototype:

```
ORM_Status
ORM_Dump(     ORM_AppNodeRef  subtree,
              long            depth,
              long            what,
              ORM_String      buffer,
              long            *maxlen
              );
```

Parameters:

subtree	The root of the subtree to dump. All navigation information is saved relative to this node.
depth	The depth, up to which entities in this subtree should be extracted. A depth of *0* means, direct childs of the given sub-root only, i.e. if the subtree points to a component node with an attribute node as one of its direct children, the name of the attribute would be extracted, but not the value or other extended attribute information, if depth=0. A depth of -1 extracts the whole subtree, independent of its depth.
what	Is a bitmask, defining what kind of entities should be extracted: ORM_DumpSetObjects ORM_DumpSetComponents ORM_DumpSetAttributes ORM_DumpSetWritable ORM_DumpSetDefault = Objects | Components | Writable ORM_DumpSetEveryThing = Objects | Components | Attributes
buffer	The address of a character buffer, where to store the extracted entity information
maxlen	Pointer to the maximum length of this buffer. On return, maxlen will contain the number of bytes used in this buffer including the C-String '\0' terminator.

3.6.3 ORM_Restore

Function to reload the saved ORM information into an existing subtree, where at least the root of the given subtree has to exist. *Note:* Because the restore request may fail with some attribute modifications already performed, an application may want to call ORM_Dump

(into a temporary buffer) before actually calling ORM_Restore, to be able to *undo* the partial operations.

Prototype:

```
ORM_Status
ORM_Restore(    ORM_ApplHandler  subtree,
                ORM_String       buffer,
                long             *length
                );
```

Parameters:

subtree	Node of the subtree to load the management information into.
buffer	pointer to ORM subrequest sequence.
length	pointer to length of the request. On return, this will contain the number of bytes processed from this request.

3.7 ORM SSL Generic Datatypes

```
#ifndef _ORM_TYPE_H
#define _ORM_TYPE_H

/*
 * Some generic definitions, may become obsolete
 */

typedef enum {
    False,
    True
    } boolean;

#ifndef NULL
#define NULL (void *)0
#endif

typedef unsigned long size_t;

#define ORM_Ptr( base, offset) (void *)((size_t)(base)+(size_t)(offset))

#define ORM_Malloc(x)    (void *)malloc(x)
#define ORM_Free(x)      free(x)

/*
 * more ORM specific stuff
 */

/*
 * How requests and responses are passed to the ORM protocol layer
 */

typedef char     *ORM_RequestDef;

typedef char     *ORM_ResponseDef;

/*
 * The principal type of every ORM protocol entity, e.g. names and values,
 * but also used most C-strings.
 */

typedef char     *ORM_String;

/*
 * Used for StateMaps and String Maps as the lookup key
 */

typedef long     ORM_Key;


/*
 * The following are various opaque handles. Opaque mainly to the protocol
 * layer but also for the lower of two stacked layers.
 */

typedef void     *ORM_AppNodeDef;
typedef void     *ORM_AppHandleDef;
typedef void     *ORM_AppAspectDef;
typedef void     *ORM_AppDataPtrDef;
```

```
typedef void     *ORM_AppAttribDescrDef;
typedef void     *ORM_AppCallContextDef;
typedef void     *ORM_AppConverterArgDef;

/*
 * Valid Access modes for an attribute
 */

typedef enum {
    ORM_AttribModeNone,
    ORM_AttribModeRW,
    ORM_AttribModeRO,
    ORM_AttribModeWO,
    ORM_AttribModeRWP
    } ORM_AttribModeDef;

/*
 * Known (native) datatypes, which are supported by the Generic converter
 */

typedef enum {
    ORM_AttribTypeNone,
    ORM_AttribTypeInt1,
    ORM_AttribTypeUInt1,
    ORM_AttribTypeInt2,
    ORM_AttribTypeUInt2,
    ORM_AttribTypeInt4,
    ORM_AttribTypeUInt4,
    ORM_AttribTypeInt8,
    ORM_AttribTypeUInt8,
    ORM_AttribTypeReal32,
    ORM_AttribTypeReal64,
    ORM_AttribTypeString,
    ORM_AttribTypeHexDat,
    ORM_AttribTypeSelect,     /* 1 out of many */
    ORM_AttribTypeState,      /* 1 out of many, but with dynamic range */
    ORM_AttribTypeOption,     /* binary switch ON/OFF YES/NO */
    ORM_AttribTypeMChoice,    /* n out of many */
    ORM_AttribTypeUnknown
    } ORM_AttribTypeDef;

/*
 * ORM Error Codes, used as well by the protocol as by the ORM SSL
 */

typedef enum {
   ORM_ENoError,              /* Operation successfull! */
   ORM_EPermission,           /* None or wrong auth.information */
   ORM_ENoSuchNode,           /* some name in pathname could not be found */
   ORM_ENoSuchAttribute,      /* Attribute in Set-Request doesn't exist */
   ORM_ENoSuchObject,         /* Object/Manager could not be found */
   ORM_EInvalidOperation,     /* Operation not applicable to node type */
   ORM_EProtocol,             /* ORM protocol violation */
   ORM_ECommunication,        /* lower level comm error */
   ORM_ERange,                /* new attribute value out of range */
   ORM_EParameterList,        /* set of attributes not applicable */
   ORM_EMissingAttribute,     /* mandatory attrib. missing or NULL */
   ORM_ENoSpace,              /* internal allocation */
   ORM_ENoBuffer,             /* response buffer */
```

```
    ORM_EInternal,              /* ORM Internal error -> bug */
    ORM_EApplication,           /* application level error -> bug */
    } ORM_Status;


/*
 * Types of nodes in the virtual tree. Note, that only nodes of type
 * ORM_NodeTypeComponent can have children!
 */

typedef enum {
    ORM_NodeTypeUnknown,
    ORM_NodeTypeObject,
    ORM_NodeTypeComponent,
    ORM_NodeTypeAttribute,
    ORM_NodeTypeAny
    } ORM_NodeTypeDef;

/*
 * Types of ORM requests. Note that the Dump and Restore requests are
 * not part of the ORM protocol, but only available within the ORM
 * server support library
 */

typedef enum {
    ORM_RequestObjectGet,
    ORM_RequestComponentGet,
    ORM_RequestAttributeGet,
    ORM_RequestAttributeInfoGet,
    ORM_RequestAttributeSet,
    ORM_RequestSet,
    ORM_RequestGet,
    ORM_RequestDump,
    ORM_RequestRestore
    } ORM_RequestTypeDef;


#endif
```

WHAT IS CLAIMED IS:

1. A system for managing objects, including a first server, comprising:

a first receiver portion configured to receive a request in a hypermedia format;

a first translator portion configured to convert the hypermedia request to an object request;

a sender portion configured to send the object request to an object manager;

a second receiver portion configured to receive a response from the object manager; and

a second translator portion configured to convert the object manager response to the hypermedia format.

2. The system of claim 1, further comprising a second server, including:

a third receiver portion configured to receive a request in a hypermedia format;

a third translator portion configured to convert the hypermedia request to an object request;

a second sender portion configured to send the object request to an object manager;

a fourth receiver portion configured to receive a response from the object manager; and

a fourth translator portion configured to convert the object manager response to the hypermedia format.

3. The system of claim 1, further comprising:

a second sending portion configured to send the hypermedia format data from the sender portion to a browser to be displayed.

4. The system of claim 1, where the object manager manages a self-describing object.

5. The system of claim 1, where the object manager manages a non-self-describing object.

6. The system of claim 5, where the object manager performs a "worm" function.

7. A method for browsing objects, where a browser communicates with a server, comprising the steps, performed by the browser, of:

sending an initial URL to the server;

receiving first data from the server, where the first data specifies an object corresponding to the URL;

sending user-entered data associated with the object to the server; and

receiving second data from the server, where the second data specifies a second object corresponding to the user-entered data.

8. The method of claim 7,

wherein the step of sending an initial URL to the server comprises the step of sending an initial URL known to the browser, where the URL is the URL of the server.

9. The method of claim 7,

wherein the step of sending an initial URL to the server comprises the step of sending an initial URL entered by the user, where the URL is the URL of the server.

10. The method of claim 7,

wherein the step of sending user-entered data associated with the object to the server includes the step of indicating a "set" operation in the user-entered data.

11. The method of claim 7,

wherein the step of sending user-entered data associated with the object to the server includes the step of indicating a "get" operation in the user-entered data.

12. The method of claim 7, wherein the step of receiving second data from the server includes the step of receiving data corresponding to an attribute value of the object.

13. The method of claim 7, wherein the step of receiving second data from the server includes the step of receiving data corresponding to a second object linked to the first object via an object-link.

14. A computer program product comprising:

a computer usable medium having computer readable code embodied therein for managing objects, the computer program product comprising:

computer readable program code devices configured to cause a computer to effect receiving a request in a hypermedia format;

computer readable program code devices configured to cause a computer to effect converting the hypermedia request to an object request;

computer readable program code devices configured to cause a computer to effect sending the object request to an object manager;

computer readable program code devices configured to cause a computer to effect receiving a response from the object manager; and

computer readable program code devices configured to cause a computer to effect converting the object manager response to a second hypermedia format.

15. The computer program product of claim 14, further comprising:

computer readable program code devices configured to cause a computer to effect sending the second hypermedia format data to a browser to be displayed.

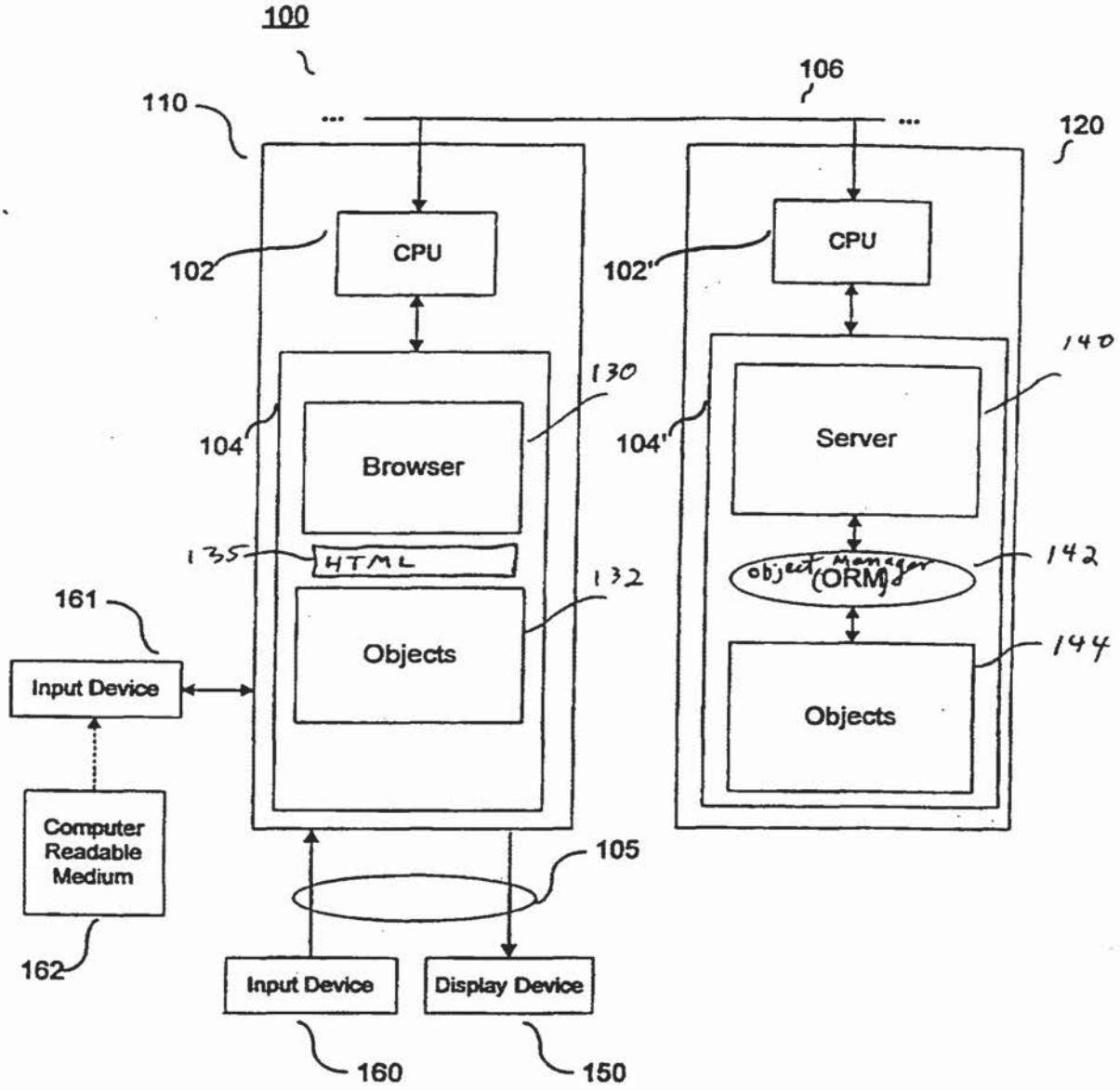


Fig. 1

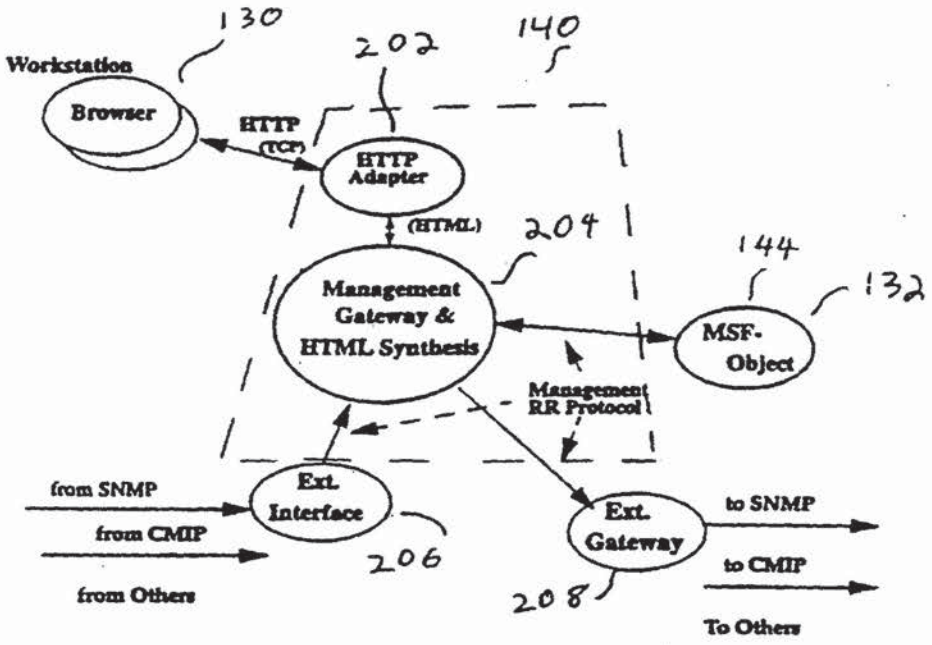


Fig. 2

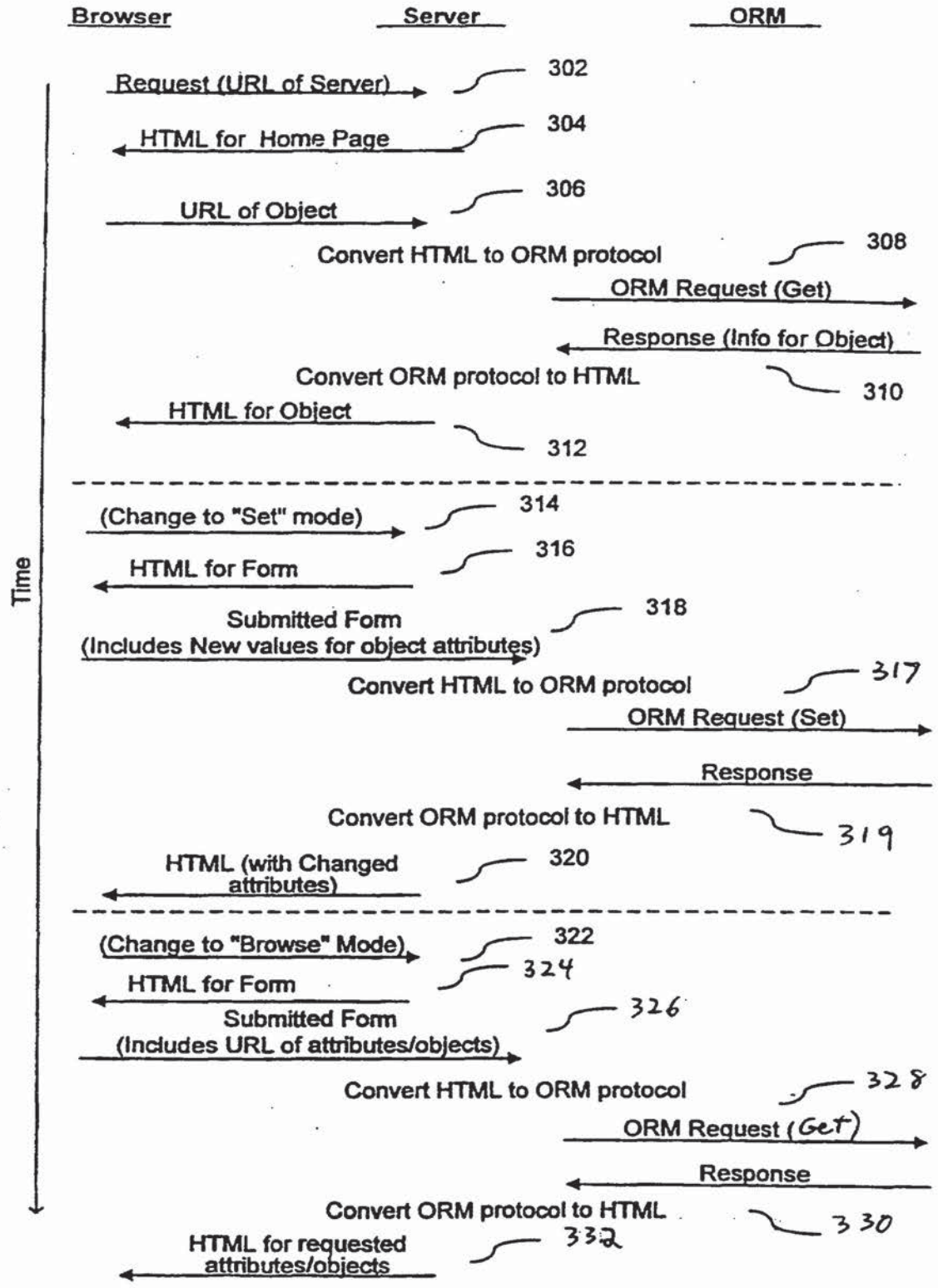


Fig. 3

A System as a Tree of Managed Entities:

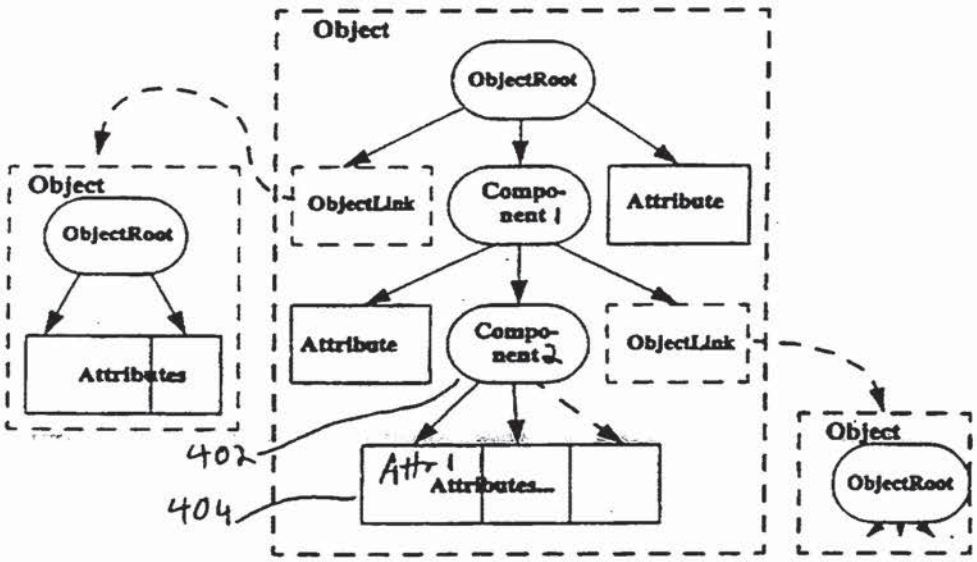


Fig. 4

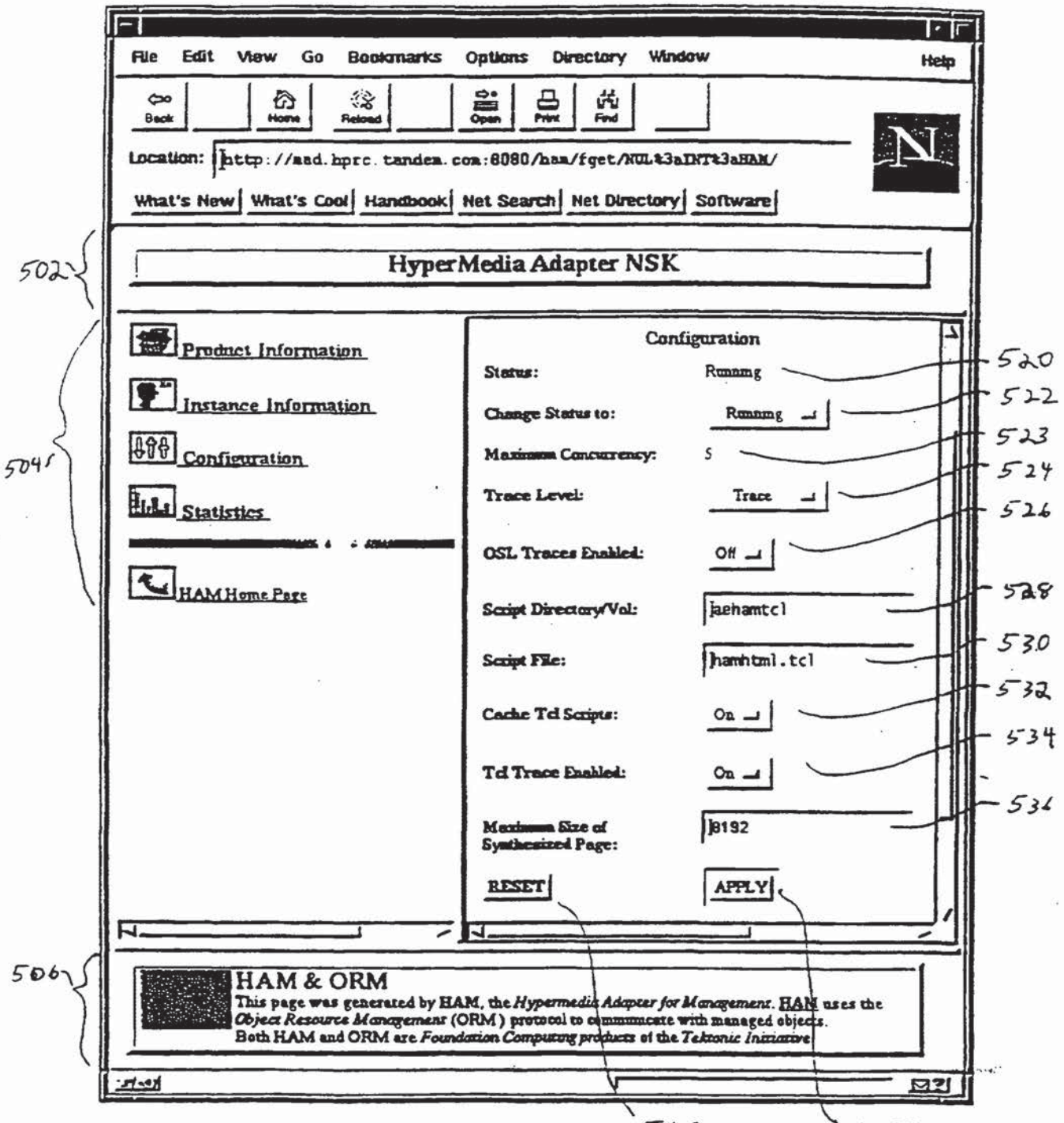


Fig. 5

hamsetup.html

```
 1 <HTML>
 2 <HEAD>
 3 <TITLE>Alter Configuration of HyperMedia Adapter NSK@TCP:168.87.28.7/8062:HAM </TI
   +TLE>
 4 <BASE HREF='http://mad.hprc.tandem.com:8080/ham/set/TCP%3a168.87.28.7%2f8062%3aHAM
   +%2fHyperMedia%20Adapter%20NSK%2fConfiguration/'>
 5 </HEAD>
 6
 7 <BODY BACKGROUND="/icons/icstgry.gif">
 8 <H2>
 9 <IMG ALT="" SRC="/icons/pccnfig.gif">
10 Alter Configuration</H2>
11 <H3>of HyperMedia Adapter NSK</H3>
12 <P><CENTER><IMG SRC="/icons/pcbrblu.gif"></CENTER><P>
13 <FORM METHOD=POST ACTION='http://mad.hprc.tandem.com:8080/ham/cset/TCP%3a168.87.28
   +.7%2f8062%3aHAM%2fHyperMedia%20Adapter%20NSK%2fConfiguration/'>
14 <TABLE >
15 <P><CAPTION><STRONG><FONT SIZE=4>Configuration</FONT></STRONG></CAPTION>
16 <TR><TH ALIGN=LEFT><P>Status:</TH>
17 <TD ALIGN=LEFT>Running</TD>
18 <TR VALIGN=BASELINE>
19 <TH ALIGN='LEFT'><P>Change Status to:</TH><TD >
20 <SELECT NAME='Change Status to'>
21 <OPTION SELECTED> Running
22 <OPTION> Suspended
23 <OPTION> Aborted
24 <OPTION> Stopped
25 </SELECT>
26 </TD></TR>
27 <TR><TH ALIGN=LEFT><P>Maximum Concurrency:</TH>
28 <TD ALIGN=LEFT>5</TD>
29 <TR VALIGN=BASELINE>
30 <TH ALIGN='LEFT'><P>Trace Level:</TH><TD >
31 <SELECT NAME='Trace Level'>
32 <OPTION SELECTED> Warnings
33 <OPTION> Errors
34 <OPTION> Info
35 <OPTION> Debug
36 <OPTION> Trace
37 <OPTION> Everything
38 </SELECT>
39 </TD></TR>
40 <TR VALIGN=BASELINE>
41 <TH ALIGN='LEFT'><P>OSL Traces Enabled:</TH><TD >
42 <SELECT NAME='OSL Traces Enabled'>
43 <OPTION SELECTED> Off
44 <OPTION> On
45 </SELECT>
46 </TD></TR>
47 <TR VALIGN=BASELINE>
48 <TH ALIGN='LEFT'><P>Script Directory/Vol:</TH><TD >
49 <INPUT NAME='Script Directory/Vol'
50 TYPE=TEXT VALUE='sehamtcl'>
51 </TD></TR>
52 <TR VALIGN=BASELINE>
53 <TH ALIGN='LEFT'><P>Script File:</TH><TD >
54 <INPUT NAME='Script File'
55 TYPE=TEXT VALUE='hamhtml.tcl'>
56 </TD></TR>
57 <TR VALIGN=BASELINE>
58 <TH ALIGN='LEFT'><P>Cache Tcl Scripts:</TH><TD >
59 <SELECT NAME='Cache Tcl Scripts'>
60 <OPTION SELECTED> On
61 <OPTION> Off
62 </SELECT>
63 </TD></TR>
64 <TR VALIGN=BASELINE>
65 <TH ALIGN='LEFT'><P>Tcl Trace Enabled:</TH><TD >
66 <SELECT NAME='Tcl Trace Enabled'>
67 <OPTION SELECTED> Off
68 <OPTION> On
69 </SELECT>
70 </TD></TR>
```

Fig 6(a)

hamsetup.html

```
71 <TR VALIGN=BASELINE>
72 <TH ALIGN='LEFT'><P>Maximum Size of Synthesized Page:</TH><TD >
73 <INPUT NAME='Maximum Size of Synthesized Page'
74 TYPE=TEXT VALUE='8192'>
75 </TD></TR>
76 <TR><TH ALIGN=LEFT><P><INPUT TYPE='reset' VALUE='RESET'></TH>
77 <TD ALIGN=LEFT><INPUT TYPE='submit' VALUE='APPLY'></TD></TR>
78 </TABLE>
79 </FORM>
80 <P>Go to <A HREF='http://mad.hprc.tandem.com:8080/ham/cget/TCP%3a168.87.28.7%2f806
   +2%3aHAM%2fHyperMedia%20Adapter%20NSK%2fConfiguration/'><STRONG>Browse-Only Mode</ST
   +RONG></A> of
81 <STRONG>Configuration</STRONG>
82 <P>
83 <P><CENTER><IMG SRC='/icons/pcbrblu.gif'></CENTER><P>
84 <BR><A HREF='http://mad.hprc.tandem.com:8080/ham/get/' TARGET='_top'>
85 <IMG SRC="/icons/icback.gif"> HAM Home Page</A>
86 </BODY></HTML>
```

Fig 6(b)

hamhead.html Page 1

```
<HTML>
<HEAD>
<TITLE>HyperMedia Adapter NSK@TCP:168.87.28.7/8062:HAM </TITLE>
<BASE HREF="http://mad.hprc.tandem.com:8080/ham/get/TCP%3a168.87.28.7%2f8062%3aHAM
+%2fHyperMedia%20Adapter%20NSK/">
</HEAD>

<BODY BACKGROUND="/icons/icstgry.gif">
<TABLE WIDTH="100%" HEIGHT="100%" ALIGN="CENTER" |
        BORDER="4">
<TR><TH><FONT SIZE=5>HyperMedia Adapter NSK</FONT></TH></TR>
</TABLE></BODY></HTML>
```

702

Fig 7(a)

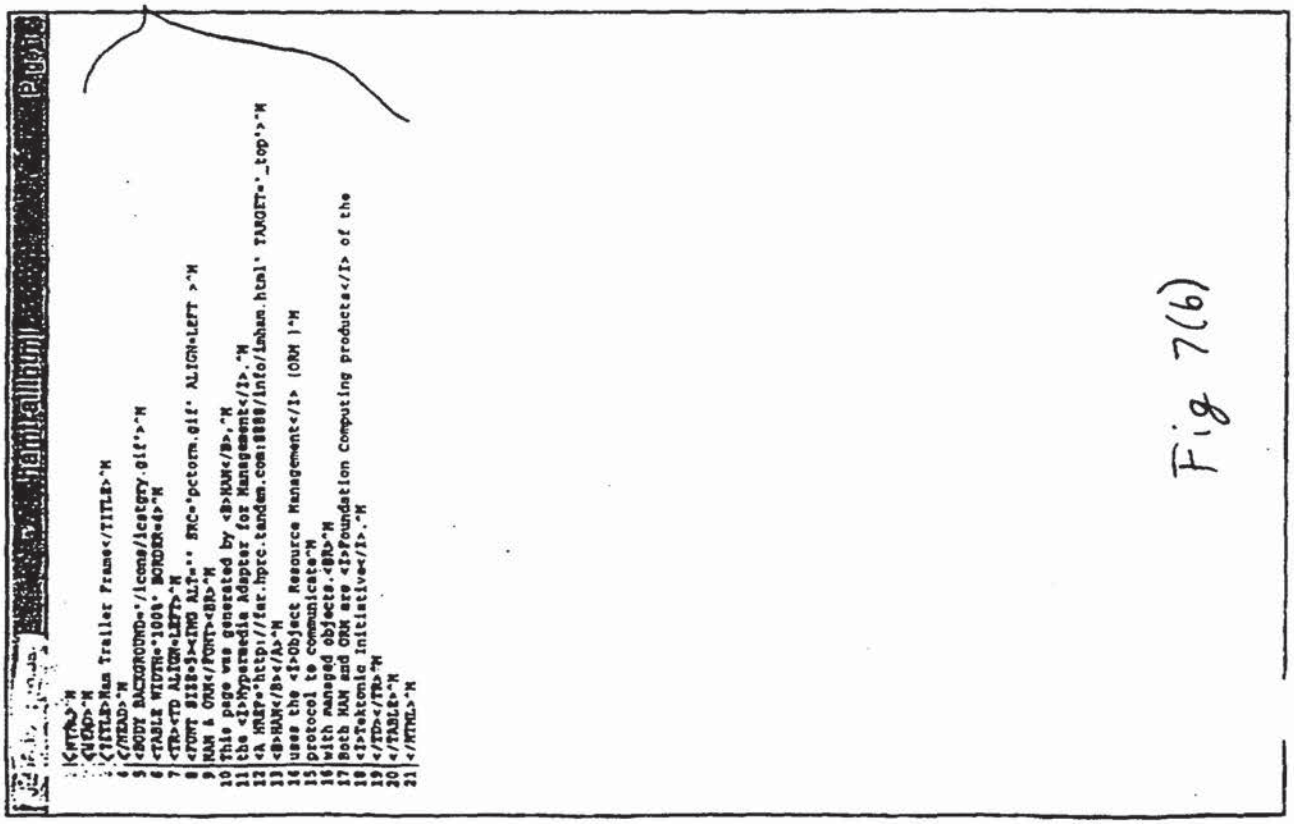


Fig 7(b)

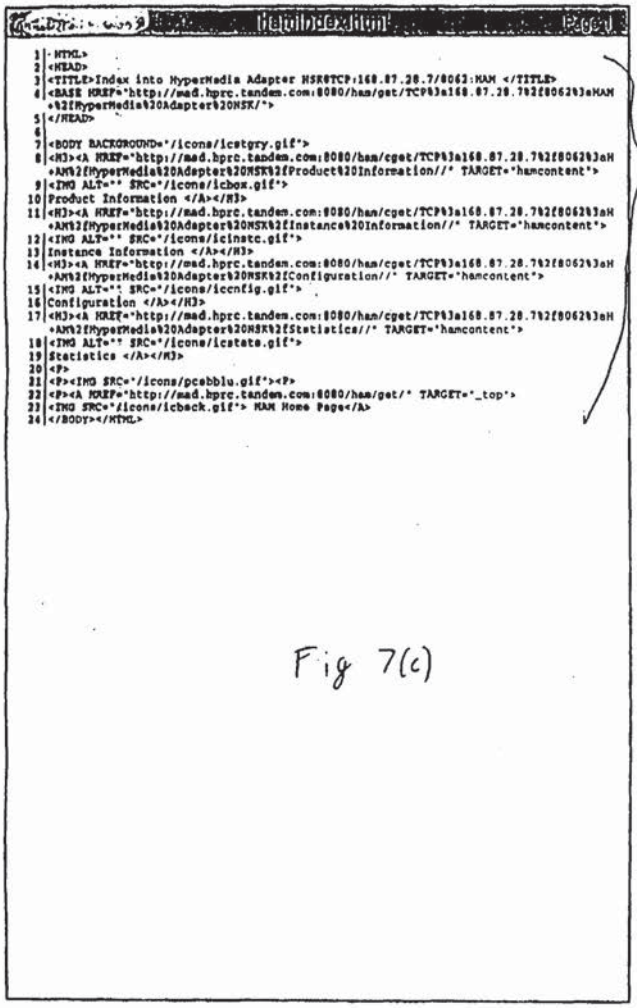


Fig 7(c)

ORM Protocol examples

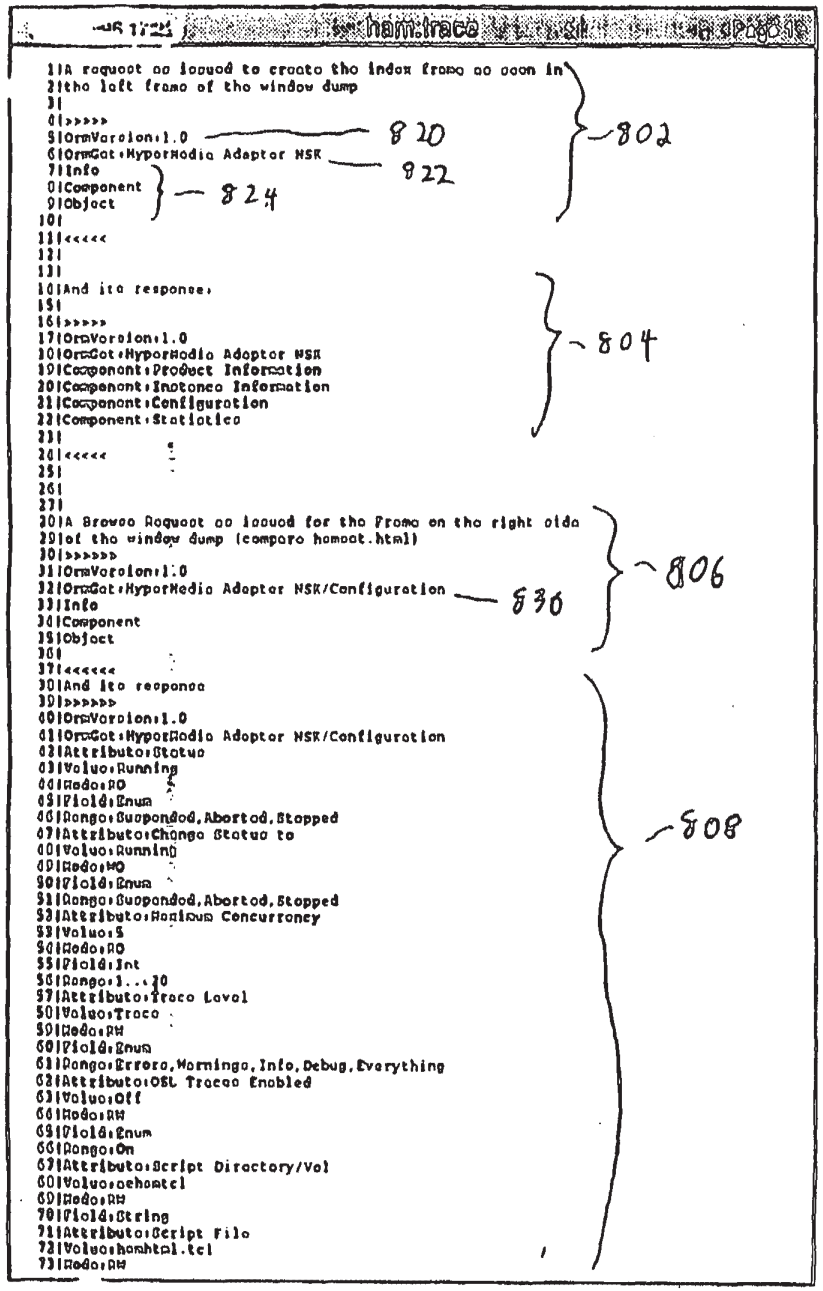


Fig. 8(a)

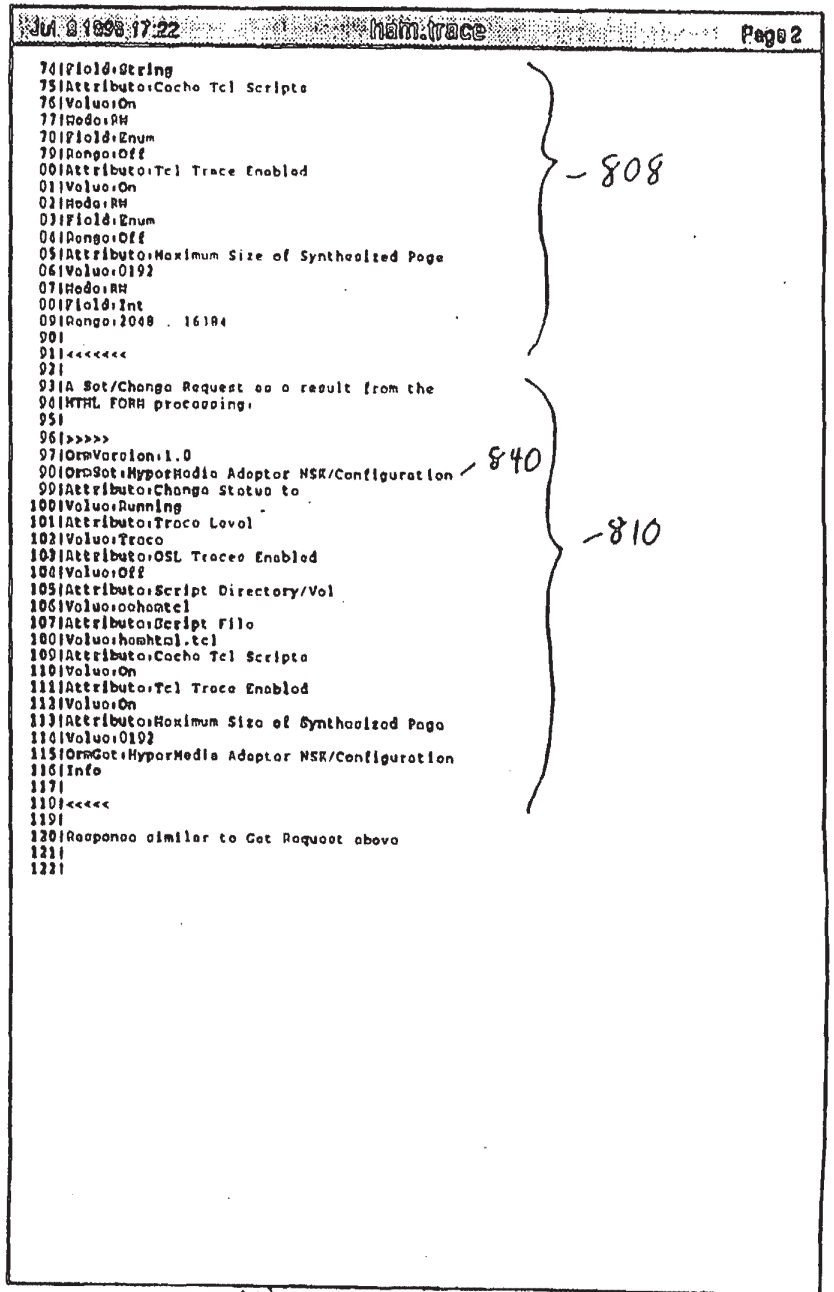


Fig. 8(b)

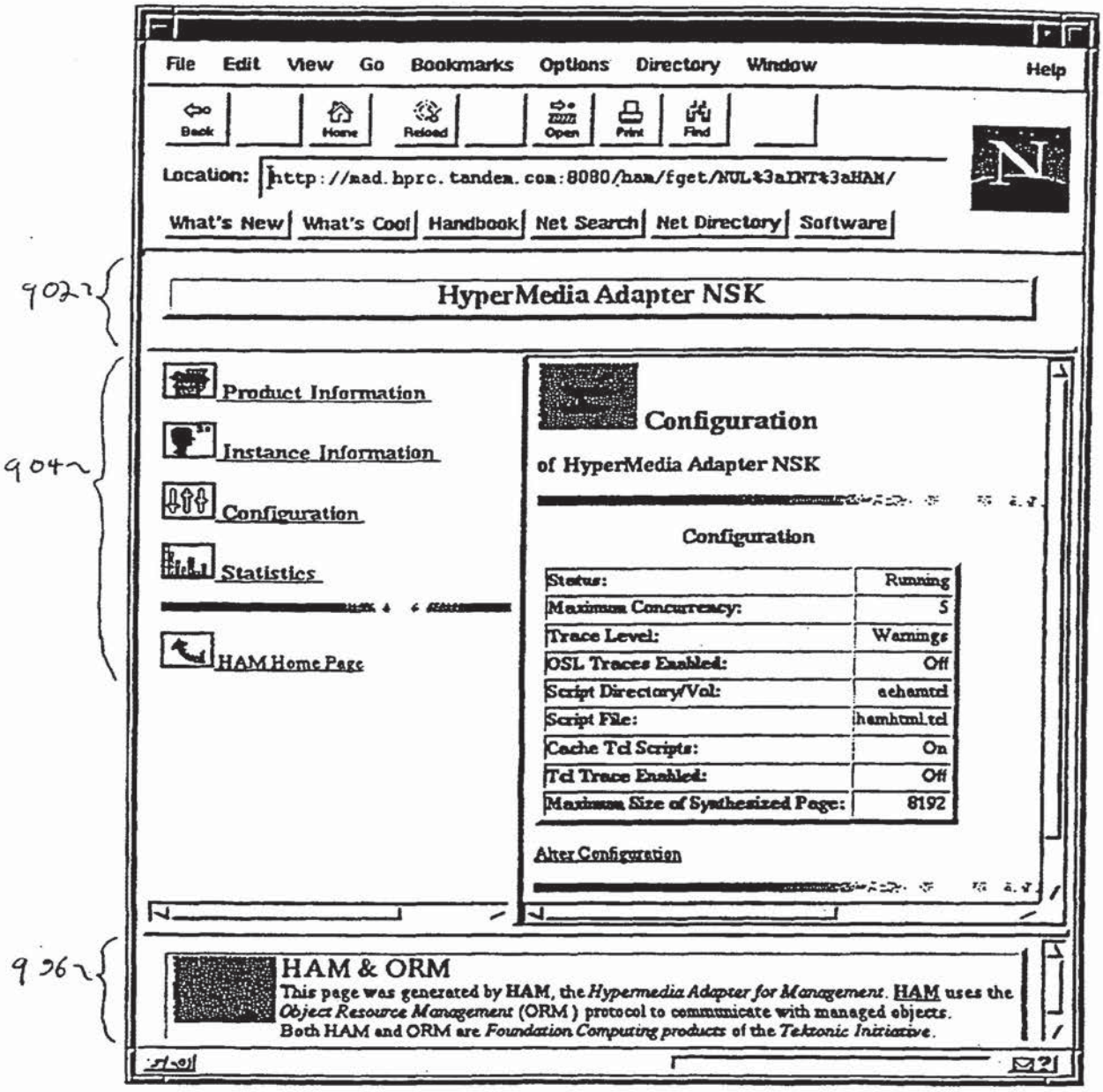
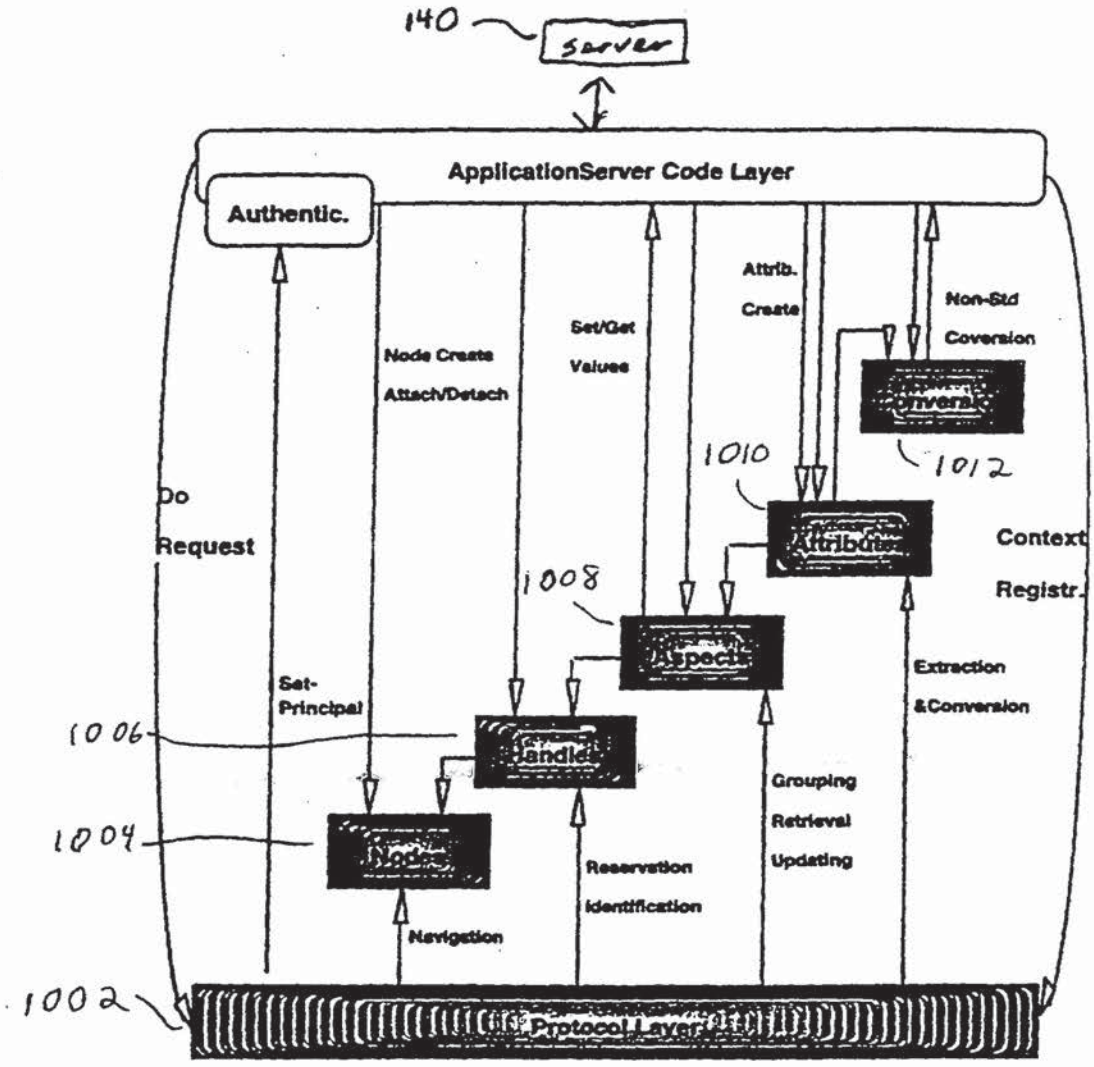


Fig. 9



The Layered Structure of the ORM Support Library

Fig. 10

INTERNATIONAL SEARCH REPORT

Inter onal Application No
PCT/US 97/11885

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	JAGANNATHAN V ET AL: "COLLABORATIVE INFRASTRUCTURES USING THE WWW AND CORBA-BASED ENVIRONMENTS" PROCEEDINGS - THE WORKSHOP ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES, 19 June 1996, pages 292-297, XP000645510 see page 293, column 1, line 39 - page 294, column 1, line 5 -----	1,14

☐ Further documents are listed in the continuation of box C.

☐ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
6 November 1997	12. 11. 97

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Katerbau, R

Form PCT/ISA/210 (second sheet) (July 1992)

BNSDOCID: <WO___9802831A1_I_>

THIS PAGE BLANK (USPTO)

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

THIS PAGE BLANK (USPTO)

PTO/SB/21 (modified)
Approved for use through xx/xx/xx, OMB 0651-0031
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

0001/PTO
Rev. 10/95

U.S. Department of Commerce
Patent and Trademark Office

TRANSMITTAL FORM

(to be used for all correspondence during pendency of filed application)

Application Number	09/284,113
Filing Date	April 7, 1999
First Named Inventor	Michael De Angelo
Group Art Unit Number	2771
Examiner Name	not yet known
Attorney Docket Number	3726

Total Number of Pages in This Submission: 5

ENCLOSURES *(check all that apply)*

- ☐ Fee Transmittal Form (in duplicate)
  - ☐ Check Enclosed
- ☒ Return Receipt Postcard
- ☐ Response to Notice to File Missing Parts
- ☐ Assignment & Recordation Cover Sheet
- ☐ Declaration
- ☐ Small Entity Statement
- ☐ Information Disclosure Statement & PTO-1449
  - ☐ Copies of IDS Cited References
- ☒ Request for Corrected Filing Receipt
- ☐ Request for Correction of Recorded Assignment
- ☐ Amendment/Response: [ ] Page(s)
  - ☐ After Final
- ☐ Status Request
- ☐ Revocation and Power of Attorney
- ☐ Issue Fee Transmittal
- ☐ Letter to Chief Draftsperson
- ☐ Formal Drawing(s):
  - [ ] Sheet(s) of Figure(s) [ ]
- ☐ Appeal Communication to Board of Appeals and Interferences
- ☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
- ☐ Certified Copy of Priority Document(s)
- ☐ After Allowance Communication to Group
- ☒ Copy of Official Filing Receipt
- ☒ Copy of executed Verified Statement Claiming Small Entity Status
- ☐ _____
- ☐ _____
- ☐ _____
- ☐ _____

REMARKS:

SIGNATURE OF ATTORNEY OR AGENT

Signature:			
Attorney/Reg. No.:	Greg T. Sueoka / Reg. No.: 33,800	Dated:	May 8, 2000

CERTIFICATE OF MAILING

I hereby certify that this correspondence, including the enclosures identified above, is being deposited with the United States Postal Service as first class mail in an envelope addressed to: The Assistant Commissioner for Patents, Washington, D.C. 20231 on the date shown below. If the Express Mail Mailing Number is filled in below, then this correspondence is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service pursuant to 37 CFR 1.10.

Signature:			
Typed or Printed Name:	Greg T. Sueoka	Dated:	May 8, 2000
Express Mail Mailing Number *(optional)*:			

IN THE UNITED STATES

PATENT AND TRADEMARK OFFICE

APPLICANT:	Michael De Angelo
SERIAL NO.:	09/284,113
FILING DATE:	April 7, 1999
TITLE:	System And Method For Creating And Manipulating Information Containers With Dynamic Registers
EXAMINER:	not yet known
GROUP ART UNIT:	2771
ATTY. DKT. NO.:	3726

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Assistant Commissioner For Patents, Washington, DC 20231, on the date shown below:

Dated: May 8 2000 By: _____
Greg T. Sueoka, Reg. No.: 33,800

ASSISTANT COMMISSIONER FOR PATENTS
APPLICATION PROCESSING DIVISION
CUSTOMER CORRECTION BRANCH
WASHINGTON, DC. 20231

REQUEST FOR CORRECTED FILING RECEIPT

SIR:

Enclosed is a copy of the Official Filing Receipt. It contains the following error:

1. The filing receipt does not indicate small entity status, as evidenced by the executed Verified Statement Claiming Small Entity Status (37 CFR 1.9(f) & 1.27(c))—Small Business Concern, a copy of which is enclosed.

Please issue a corrected Filing Receipt rectifying this error.

☑ The correction is not due to any error by the Applicant and therefore no fee is due.

☐ Since at least one of the corrections is due to Applicant's error, payment in the amount of $25, pursuant to 37 CFR § 1.19(h), is enclosed.

Respectfully submitted,
MICHAEL DE ANGELO

Dated: May 8, 2000

By: _____

Greg T. Sueoka, Reg. No.: 33,800
Fenwick & West LLP
Two Palo Alto Square
Palo Alto, CA 94306
Tel.: (650) 858-7194
Fax.: (650) 494-1417

21114/03726/DOCS/1042815.1

FILING RECEIPT

OC000000005048164

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office

Address: ASSISTANT SECRETARY AND COMMISSIONER OF PATENT AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NUMBER	FILING DATE	GRP ART UNIT	FIL FEE REC'D	ATTY.DOCKET.NO	DRAWINGS	TOT CLAIMS	IND CLAIMS
09/284,113	04/07/1999	2771	524	3726-US	30	36	3

GREG T SUEOKA
FENWICK & WEST
TWO PALO ALTO SQUARE
PALO ALTO, CA 94306

Date Mailed: 04/13/2000

Receipt is acknowledged of this nonprovisional Patent Application. It will be considered in its order and you will be notified as to the results of the examination. Be sure to provide the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION when inquiring about this application. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. **If an error is noted on this Filing Receipt, please write to the Office of Initial Patent Examination's Customer Service Center. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the PTO processes the reply to the Notice, the PTO will generate another Filing Receipt incorporating the requested corrections (if appropriate).**

Applicant(s)

MICHAEL DE ANGELO, SANTA BARBARA, CA UNITED STATES;

Small Entity Status

Continuing Data as Claimed by Applicant

THIS APPLICATION IS A 371 OF PCT/US99/01988 01/28/1999
WHICH CLAIMS BENEFIT OF 60/073,209 01/30/1998

Foreign Applications

If Required, Foreign Filing License Granted 04/12/2000

**

Title

SYSTEM AND METHOD FOR CREATING AND MANIPULATING INFORMATION CONTAINERS WITH DYNAMIC REGISTERS

Preliminary Class

707

Data entry by : BARRETO, NGA **Team :** OIPE **Date:** 04/13/2000

PTO/SB/10 (6-95) (modified)
Approved for use through 07/31/96 OMB 0651-0031
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

VERIFIED STATEMENT CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) & ~~1.27(c)~~)–SMALL BUSINESS CONCERN	Docket Number (Optional): 3726

Applicant or Patentee: Michael De Angelo

Application or Patent No.:

Filing Date or Issue Date:

Title: System And Method For Creating And Manipulating Information Containers With Dynamic Registers

I hereby declare that I am

[ ] the owner of the small business concern identified below:

[X] an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF SMALL BUSINESS CONCERN Ematrix Corporation

ADDRESS OF SMALL BUSINESS CONCERN 104 West Anapamu, Suite C
Santa Barbara, California 93101

I hereby declare that the above identified small business concern qualifies as a small business concern as defined in 13 CFR 121.12, and reproduced in 37 CFR 1.9(d), for purposes of paying reduced fees to the United States Patent and Trademark Office, in that the number of employees of the concern, including those of its affiliates, does not exceed 500 persons. For purposes of this statement, (1) the number of employees of the business concern is the average over the previous fiscal year of the concern of the persons employed on a full-time, part-time or temporary basis during each of the pay periods of the fiscal year, and (2) concerns are affiliates of each other when either, directly or indirectly, one concern controls or has the power to control the other, or a third party or parties controls or has the power to control both.

I hereby declare that rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the invention described in:

[X] the specification filed herewith with title as listed above.

[ ] the application identified above.

[ ] the patent identified above.

If the rights held by the above identified small business concern are not exclusive, each individual, concern or organization having rights in the invention must file separate verified statements averring to their status as small entities, and no rights to the invention are held by any person, other than the inventor, who would not qualify as an independent inventor under 37CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d), or a nonprofit organization under 37 CFR 1.9(e).

Each such person, concern or organization having any rights in the invention is listed below:

[X] No such person, concern, or organization exists.

[ ] Each such person, concern or organization is listed below:

Separate verified statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

NAME OF PERSON SIGNING Michael De Angelo

TITLE OF PERSON IF OTHER THAN OWNER Officer

ADDRESS OF PERSON SIGNING 104 West Anapamu, Suite C, Santa Barbara, California 93101

SIGNATURE Michael De Angelo DATE April 5, 1999

Rev. 04/01/99

09/284 113

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

U.S. APPLICATION NO.		FIRST NAMED APPLICANT	ATTY. DOCKET NO.
09/284,113		DE ANGELO M	3726 US

INTERNATIONAL APPLICATION NO
PCT/US99/01988

I.A. FILING DATE	PRIORITY DATE
01/28/99	01/30/98

5611

GREG T SUEOKA
FENWICK & WEST
TWO PALO ALTO SQUARE
PALO ALTO CA 94306

DATE MAILED 01/13/00

NOTIFICATION OF ACCEPTANCE OF APPLICATION UNDER 35 U.S.C. 371 AND 37 CFR 1.494 OR 1.495

1. The applicant is hereby advised that the United States Patent and Trademark Office in its capacity as ☑ a Designated Office (37 CFR 1.494), ☐ an Elected Office (37 CFR 1.495), has determined that the above identified international application has met the requirements of 35 U.S.C. 371, and is **ACCEPTED** for national patentability examination in the United States Patent and Trademark Office.

2. The United States Application Number assigned to the application is shown above and the relevant dates are:

April 7, 1999
35 U.S.C. 102(e) DATE

April 7, 1999
DATE OF RECEIPT OF
35 U.S.C. 371 REQUIREMENTS

A Filing Receipt (PTO-103X) will be issued for the present application in due course. **THE DATE APPEARING ON THE FILING RECEIPT AS THE "FILING DATE" IS THE DATE ON WHICH THE LAST OF THE 35 U.S.C. 371(C) REQUIREMENTS HAS BEEN RECEIVED IN THE OFFICE. THIS DATE IS SHOWN ABOVE.** *The filing date of the above identified application is the international filing date of the international application (Article 11(3) and 35 U.S.C. 363).* Once the Filing Receipt has been received, send all correspondence to the Group Art Unit designated thereon.

3. ☑ A request for immediate examination under 35 U.S.C. 371(f) was received on April 7, 1999 and the application will be examined in turn.

4. The following items have been received:
- ☑ U.S. Basic National Fee.
- ☑ Copy of the international application in:
  - ☐ a non-English language.
  - ☑ English.
- ☐ Translation of the international application into English.
- ☑ Oath or Declaration of inventors(s) for DO/EO/US.
- ☐ Copy of Article 19 amendments. ☐ Translation of Article 19 amendments into English.
  The Article 19 amendments ☐ have ☐ have not been entered.
- ☐ The International Preliminary Examination Report in English and its Annexes, if any.
- ☐ Copy of the Annexes to the International Preliminary Examination Report (IPER).
  - ☐ Translation of Annexes to the IPER into English.

  The Annexes ☐ have ☐ have not been entered.
- ☐ Preliminary amendment(s) filed _____ and _____.
- ☐ Information Disclosure Statement(s) filed _____ and _____.
- ☑ Assignment document.
- ☑ Power of Attorney and/or Change of Address.
- ☐ Substitute specification filed _____.
- ☑ Statement Claiming Small Entity Status.
- ☐ Priority Document.
- ☑ Copy of the International Search Report ☐ and copies of the references cited therein.
- ☐ Other:

Applicant is reminded that any communication to the United States Patent and Trademark Office must be mailed to the address given in the heading and include the U.S. application no. shown above. (37 CFR 1.5)

Barbara Campbell
National Stage Processing
Telephone: (703) (703) 305-3631

FORM PCT/DO/EO/903 (December 1997)

IN THE UNITED STATES

PATENT AND TRADEMARK OFFICE

APPLICANT:	Michael De Angelo
SERIAL NO.:	09/284,113
FILING DATE:	April 7, 1999
TITLE:	System And Method For Creating And Manipulating Information Containers With Dynamic Registers
EXAMINER:	Unknown
GROUP ART UNIT:	Unknown
ATTY. DKT. NO.:	3726

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Assistant Commissioner For Patents, Washington, D.C. 20231, on the date shown below:

Dated: July 7, 1999 By: _____
Greg T. Sueoka, Reg. No.: 33,800

ASSISTANT COMMISSIONER FOR PATENTS
WASHINGTON, DC. 20231

INFORMATION DISCLOSURE STATEMENT
Under 37 CFR §§ 1.56 and 1.97-98

SIR:

Pursuant to the provisions of 37 CFR §§ 1.56 and 1.97-98, enclosed herewith is modified form PTO-1449 listing references for consideration by the Examiner. A copy is enclosed herewith of each listed reference which may be material to the examination of this application, and with respect to which there may be a duty to disclose.

The filing of this Information Disclosure Statement shall not be construed as a representation regarding the completeness of the list of references, or that inclusion of a reference in this list is an admission that it is prior art or is pertinent to this application, or that a search has been made, or as an admission that the information listed is, or may be considered to be, material to patentability, or that no other material information exists, and shall not be construed as an admission against interest in any manner.

☐ This application relies, under 35 U.S.C. § 120, on the earlier filing date of prior application Serial No. [SERIAL NUMBER], filed on [FILING DATE], and the references cited therein are hereby referenced, but are not required to be provided in this application under 37 CFR § 1.98(d).

The Information Disclosure Statement submitted herewith is being filed:

☐ within three months of the filing date of the application, or date of entry into the national stage of an international application, or before the mailing date of a first official action on the merits, whichever event last occurred; OR

☐ after three months of the filing date of this national application or the date of entry of the national stage in an international application, or after the mailing date of the first official action on the merits, whichever event last occurred, but before the mailing date of the first to occur of either:

(1) a final action under 37 CFR §1.113; OR

(2) a notice of allowance under 37 CFR §1.311; AND

☐ attached hereto is the fee of $240, as set forth under 37 CFR §1.17(p), for submission of this Information Disclosure Statement under 37 CFR.§ 1.97(c); OR

☑ Applicant certifies pursuant to 37 CFR § 1.97(e) that:

☑ each item of information contained in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this Statement; OR

☐ no item of information contained in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application or, to the knowledge of the person signing this certification after making reasonable inquiry, was known to any individual designated under 37 CFR § 1.56(c) more than three months prior to the filing of this Statement.

OR

☐ before the payment of the issue fee but after the mailing date of the first to occur of either:

[1] a final action under 37 CFR § 1.113; OR

[2] a notice of allowance under 37 CFR § 1.311; AND

in accordance with the requirements of 37 CFR § 1.97(d):

☐ Applicant certifies pursuant to 37 CFR. § 1.97(e) that:

- ☐ each item of information contained in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this Statement; OR
- ☐ no item of information contained in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application or, to the knowledge of the person signing this certification after making reasonable inquiry, was known to any individual designated under 37 CFR § 1.56(c) more than three months prior to the filing of this Statement; AND

☐ Applicant hereby respectfully petitions for the consideration of the accompanying Information Disclosure Statement under 37 CFR § 1.97(d)(2); AND

☐ Applicant submits the petition fee of $130 as set forth in 37 CFR § 1.17(i).

☑ Applicant submits that no fee is required for the consideration of the accompanying Information Disclosure Statement.

Consideration of the listed references and favorable action are solicited.

Respectfully submitted,
MICHAEL DE ANGELO

Dated: July 7, 1999

By: _____

Greg T. Sueoka, Reg. No.: 33,800
Fenwick & West LLP
Two Palo Alto Square
Palo Alto, CA 94306
Tel.: (650) 858-7194
Fax.: (650) 494-1417

PTO/SB/21 (Modified)
Approved for use through xx/xx/xx, OMB 0651-0031
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

0001/PTO
Rev. 10/95

U.S. Department of Commerce
Patent and Trademark Office

TRANSMITTAL FORM

(to be used for all correspondence during pendency of filed application)

Application Number	09/284,113
Filing Date	April 7, 1999
First Named Inventor	Michael De Angelo
Group Art Unit Number	Unknown
Examiner Name	Unknown
Total Number of Pages in This Submission	*5
Attorney Docket Number	3726

ENCLOSURES *(check all that apply)*

- ☐ Fee Transmittal Form (in duplicate)
  - ☐ Check Enclosed
- ☑ Return Receipt Postcard
- ☐ Response to Notice to File Missing Parts
- ☐ Assignment & Recordation Cover Sheet
- ☐ Declaration
- ☐ Small Entity Statement
- ☑ Information Disclosure Statement & PTO-1449
  - ☑ Copies of IDS Cited References
- ☐ Request for Corrected Filing Receipt
- ☐ Request for Correction of Recorded Assignment
- ☐ Amendment/Response: [ ] Page(s)
  - ☐ After Final
- ☐ Status Request
- ☐ Revocation and Power of Attorney
- ☐ Issue Fee Transmittal
- ☐ Letter to Chief Draftsperson
- ☐ Formal Drawing(s):
  [ ] Sheet(s) of Figure(s) [ ]
- ☐ Appeal Communication to Board of Appeals and Interferences
- ☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
- ☐ Certified Copy of Priority Document(s)
- ☐ After Allowance Communication to Group
- ☐ _____
- ☐ _____
- ☐ _____
- ☐ _____
- ☐ _____
- ☐ _____

REMARKS: *Does not include total pages of cited references

SIGNATURE OF ATTORNEY OR AGENT

Signature:			
Attorney/Reg. No.:	Greg T. Sueoka / Reg. No.: 33,800	Dated:	July 7, 1999

CERTIFICATE OF MAILING

I hereby certify that this correspondence, including the enclosures identified above, is being deposited with the United States Postal Service as first class mail in an envelope addressed to: The Assistant Commissioner for Patents, Washington, D.C. 20231 on the date shown below. If the Express Mail Mailing Number is filled in below, then this correspondence is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service pursuant to 37 CFR 1.10.

Signature:			
Typed or Printed Name:	Greg T. Sueoka	Dated:	July 7, 1999
Express Mail Mailing Number *(optional)*:			

Rev. 07/06/99

FORM PTO-1390 (REV. 1-98) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE	ATTORNEY'S DOCKET NUMBER 3726 US
TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371	U.S. APPLICATION NO. (If known, see 37 CFR 1.5) 09/284113 Not Yet Known

INTERNATIONAL APPLICATION NO.	INTERNATIONAL FILING DATE	PRIORITY DATE CLAIMED
PCT/US99/01988	28 January 1999	30 January 1998

TITLE OF INVENTION
System And Method For Creating And Manipulating Information Containers With Dynamic Registers

APPLICANT(S) FOR DO/EO/US
Michael De Angelo

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☑ This is a FIRST submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a SECOND or SUBSEQUENT submission of items concerning a filing under 35 U.S.C. 371.
3. ☑ This express request to begin national examination procedures (35 U.S.C. 371 (f)) at any time rather than delay examination until the expiration of the applicable time limit set in 35 U.S.C. 371(b) and PCT Articles 22 and 39(1).
4. ☐ A proper Demand for International Preliminary Examination was made by the 19th month from the earliest claimed priority date.
5. ☑ A copy of the International Application as filed (35 U.S.C. 371(c)(2)).
   a. ☐ is transmitted herewith (required only if not transmitted by the International Bureau).
   b. ☐ has been transmitted by the International Bureau.
   c. ☑ is not required, as the application was filed in the United States Receiving Office (RO/US).
6. ☐ A translation of the International Application into English (35 U.S.C. 371(c)(2)).
7. ☑ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371(c)(3)).
   a. ☐ are transmitted herewith (required only if not transmitted by the International Bureau).
   b. ☐ have been transmitted by the International Bureau.
   c. ☑ have not been made; however, the time limit for making such amendments has NOT expired.
   d. ☐ have not been made and will not be made.
8. ☐ A translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371(c)(3)).
9. ☑ An oath or declaration of the inventor(s) (35 U.S.C. 371(c)(4)).
10. ☐ A translation of the annexes of the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

Items 11. to 16. below concern document(s) or information included:

11. ☐ An Information Disclosure Statement under 37 CFR 1.97 and 1.9
12. ☑ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
13. ☐ A FIRST preliminary amendment.
    ☐ A SECOND or SUBSEQUENT preliminary amendment.
14. ☐ A substitute specification.
15. ☐ A change of power of attorney and/or address letter.
16. ☑ Other items or information: A Verified Statement Claiming Small Entity Status

U.S. APPLICATION NO. (if known. see 37 CFR 1.5)	INTERNATIONAL APPLICATION NO.	ATTORNEY'S DOCKET NUMBER
Not Yet Known	PCT/US99/01988	3726 US

17. ☑ The following fees are submitted:

BASIC NATIONAL FEE (37 CFR 1.492(a)(1)-(5)):

Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO.....$970.00

International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO..$840.00

International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO.....$760.00

International preliminary examination fee (37 CFR 1.482) paid to USPTO but all claims did not satisfy provisions of PCT Article 33(2)-(4).....$670.00

International preliminary examination fee (37 CFR 1.482) paid to USPTO and all claims satisfied provisions of PCT Article 33(2)-(4).....$96.00

	CALCULATIONS PTO USE ONLY	
ENTER APPROPRIATE BASIC FEE AMOUNT =	$760.00	
Surcharge of $130.00 for furnishing the oath or declaration later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(e)).	$0	

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE		
Total claims	36 - 20 =	16	x $18.00	$288.00	
Independent claims	3 - 3 =	0	x $78.00	$0	
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ $260.00	$0	
TOTAL OF ABOVE CALCULATIONS =				$1048.00	
Reduction of 1/2 for filing by small entity, if applicable. A Small Entity Statement must also be filed (Note 37 CFR 1.9, 1.27, 1.28). +				$524.00	
SUBTOTAL =				$524.00	
Processing fee of $130.00 for furnishing the English translation later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(f)).				$0	
TOTAL NATIONAL FEE =				$524.00	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). $40.00 per property +				$40.00	
TOTAL FEES ENCLOSED =				$564.00	
				Amount to be rendered:	$564.00
				charged:	$

a. ☑ A check in the amount of $ 564.00 to cover the above fees is enclosed.

b. ☐ Please charge my Deposit Account No. _____ in the amount of $ _____ to cover the above fees. A duplicate copy of this sheet is enclosed.

c. ☑ The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. 19-2555. A duplicate copy of this sheet is enclosed.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137 (a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

Greg T. Sueoka
FENWICK & WEST LLP

Two Palo Alto Square
Palo Alto, CA 94306

SIGNATURE

Greg T. Sueoka
NAME

33,800
REGISTRATION NUMBER

COPY

U.S. APPLICATION NO. (if known. see 37 CFR 1.5) Not Yet Known	INTERNATIONAL APPLICATION NO. PCT/US99/01988	ATTORNEY'S DOCKET NUMBER 3726 US

17. ☑ The following fees are submitted:

CALCULATIONS PTO USE ONLY

BASIC NATIONAL FEE (37 CFR 1.492(a)(1)-(5)):

Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO.....$970.00

International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO..$840.00

International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO.....$760.00

International preliminary examination fee (37 CFR 1.482) paid to USPTO but all claims did not satisfy provisions of PCT Article 33(2)-(4).....$670.00

International preliminary examination fee (37 CFR 1.482) paid to USPTO and all claims satisfied provisions of PCT Article 33(2)-(4).....$96.00

ENTER APPROPRIATE BASIC FEE AMOUNT =				$760.00	
Surcharge of $130.00 for furnishing the oath or declaration later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(e)).				$0	
CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE		
Total claims	36 - 20 =	16	x $18.00	$288.00	
Independent claims	3 - 3 =	0	x $78.00	$0	
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ $260.00	$0	
TOTAL OF ABOVE CALCULATIONS =				$1048.00	
Reduction of 1/2 for filing by small entity, if applicable. A Small Entity Statement must also be filed (Note 37 CFR 1.9, 1.27, 1.28). +				$524.00	
SUBTOTAL =				$524.00	
Processing fee of $130.00 for furnishing the English translation later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(f)).				$0	
TOTAL NATIONAL FEE =				$524.00	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). $40.00 per property +				$40.00	
TOTAL FEES ENCLOSED =				$564.00	
				Amount to be rendered:	$564.00
				charged:	$

a. ☑ A check in the amount of $ 564.00 to cover the above fees is enclosed.

b. ☐ Please charge my Deposit Account No. _____ in the amount of $ _____ to cover the above fees. A duplicate copy of this sheet is enclosed.

c. ☑ The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. 19-2555. A duplicate copy of this sheet is enclosed.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137 (a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

Greg T. Sueoka
FENWICK & WEST LLP

Two Palo Alto Square
Palo Alto, CA 94306

SIGNATURE

Greg T. Sueoka
NAME

33,800
REGISTRATION NUMBER

PTO/SB/ 10 (6-95) (modified)
Approved for use through 07/31/96. OMB 0651-0031
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

VERIFIED STATEMENT CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) & 1.27(c))--SMALL BUSINESS CONCERN	Docket Number (Optional): 3726

Applicant or Patentee: Michael De Angelo

Application or Patent No.:

Filing Date or Issue Date:

Title: **System And Method For Creating And Manipulating Information Containers With Dynamic Registers**

I hereby declare that I am

[ ] the owner of the small business concern identified below:

[X] an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF SMALL BUSINESS CONCERN Ematrix Corporation

ADDRESS OF SMALL BUSINESS CONCERN 104 West Anapamu, Suite C

Santa Barbara, California 93101

I hereby declare that the above identified small business concern qualifies as a small business concern as defined in 13 CFR 121.12, and reproduced in 37 CFR 1.9(d), for purposes of paying reduced fees to the United States Patent and Trademark Office, in that the number of employees of the concern, including those of its affiliates, does not exceed 500 persons. For purposes of this statement, (1) the number of employees of the business concern is the average over the previous fiscal year of the concern of the persons employed on a full-time, part-time or temporary basis during each of the pay periods of the fiscal year, and (2) concerns are affiliates of each other when either, directly or indirectly, one concern controls or has the power to control the other, or a third party or parties controls or has the power to control both.

I hereby declare that rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the invention described in:

[X] the specification filed herewith with title as listed above.

[ ] the application identified above.

[ ] the patent identified above.

If the rights held by the above identified small business concern are not exclusive, each individual, concern or organization having rights in the invention must file separate verified statements averring to their status as small entities, and no rights to the invention are held by any person, other than the inventor, who would not qualify as an independent inventor under 37CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d), or a nonprofit organization under 37 CFR 1.9(e).

Each such person, concern or organization having any rights in the invention is listed below:

[X] No such person, concern, or organization exists.

[ ] Each such person, concern or organization is listed below:

Separate verified statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

NAME OF PERSON SIGNING Michael De Angelo

TITLE OF PERSON IF OTHER THAN OWNER Officer

ADDRESS OF PERSON SIGNING 104 West Anapamu, Suite C, Santa Barbara, California 93101

SIGNATURE Michael De Angelo DATE April 5, 1999

Rev. 04/01/99

SYSTEM AND METHOD FOR CREATING AND MANIPULATING INFORMATION CONTAINERS WITH DYNAMIC REGISTERS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to computer systems in a multi-user mainframe or mini computer system, a client server network, or in local, wide area or public networks, and in particular, to computer networks for creating and manipulating information containers with dynamic interactive registers in a computer, media or publishing network, in order to manufacture information on, upgrade the utility of, and develop intelligence in, a computer network by offering the means to create and manipulate information containers with dynamic registers.

2. Description of the Related Art

In the present day, querying and usage of information resources on a computer network is accomplished by individuals directing a search effort by submitting key words or phrases to be compared to those key words or phrases contained in the content or description of that information resource, with indices and contents residing in a fixed location unchanging except by human input. Similarly, the class of storage medium upon which information resides, it class and subclass organizational structures, and its routes of access all remain fundamentally unaltered by ongoing user queries and usage. Only the direct and intended intervention of the owner of the information content or computer hosting site changes these parameters, normally accomplished manually by programmers or systems operators at their own discretion or the discretion of the site owner.

There exists currently in the art a limited means of interfacing a computer user with the information available on computer networks such as the world wide web. Primarily, these means are search engines. Search engines query thousands or tens of thousands of index pages per second to suggest the location of information while the user waits. While factual information can be accessed, the more complex, particular or subtle the inquiry, the more branches and sub-branches need to be explored in a time consuming fashion in order to have any chance of success. Further, there are no such automatic devices that reconstruct the information into more useful groupings or makes it more accessible according to factors

attached to the content by the content creator such as the space or time relevancy of its content, or factors attached to the content by the system's compilation and analysis of the accumulated biography of that specific content's readership.

The utility of wide area and public computer networks is thus greatly limited by the static information model and infrastructure upon which those networks operate.

One problem is that on a wide area or public network, specific content such as a document remains inert, except by the direct intervention of users, and is modified neither by patterns or history of usage on the network, or the existence of other content on the network.

Another problem is that content does not reside in an information infrastructure conducive to reconstruction by expert rule-based, fuzzy logic, or artificial intelligence based systems. Neither the intelligence of other information users nor the expert intelligence of an observant network computer system can be utilized in constructing, or re-constructing information resources. Where content resides in a fixed location and structure, "information" becomes something defined by the mind of the information provider rather than the mind of the information user, where the actual construction and utility of information exists. Information remains, like raw ore, in an unrefined state.

Another problem is that the class of storage medium upon which data resides cannot be system or user managed and altered according to the actual recorded and analyzed hierarchically graded usage of any given information resource residing on that storage medium except by statistical analysis of universal, undefined "hits" or visits to that page or site.

Another problem is that information resource groupings remain fixed on the given storage medium location according to the original installation by the resource author, not altered according to the actual recorded and analyzed hierarchically graded usage of that given information resource. Content itself remains inert, with no possibility of evolution.

A further problem with the prior art is that neither the search templates generated by those more knowledgeable in a given field of inquiry, nor the search strategies historically determined to be successful, or system-constructed according to analyses of search strategies historically determined to be successful, are available to inquiring users. A search template is here defined as one or more text phrases, graphics, video or audio bits, alone or in any defined outline or relational format designed to accomplish an inquiry. Internet or wide area network search may return dozens of briefs to a keyword or key phrase inquiry sometimes requiring the

time-consuming examination of multiple information resources or locations, with no historical relation to the success of any given search strategy.

A further problem is that there is limited means to add to, subtract from, or alter the information content of documents, databases, or sites without communicating with the owners or operators of those information resources, e.g., contacting, obtaining permission, negotiating and manually altering, adding or subtracting content. Additionally, once so altered, there is not a means to derive a proportionate value, and thereby a proportionate royalty as the information is used.

A final problem is that the physical residence of a body of data or its cyberspace location may not serve its largest body of users in the most expedient manner of access. Neither the expert intelligence of other information users nor the expert intelligence of an observant computer system is presently utilized by inherent network intelligence to analyze, re-design and construct access routes to information medium except by statistical analysis of universal, undefined "hits" or visits to that page or site.

Therefore, there is a need for a system and methods for creating and manipulating information containers with dynamic interactive registers defining more comprehensive information about contained content in a computer, media or publishing network, in order to manufacture information on, upgrade the utility of, and develop intelligence in, a computer network by providing a searching user the means to utilize the searches of other users or the historically determined and compiled searches of the system, a means to containerize information with multiple registers governing the interaction of that container, a means to re-classify the storage medium and location of information resources resident on the network, a means to allow the reconstruction of content into more useful formations, and a means to reconstruct the access routes to that information.

SUMMARY OF THE INVENTION

The present invention is a system and methods for manufacturing information on, upgrading the utility of, and developing intelligence in, a computer or digital network, local, wide area, public, corporate, or digital-based, supported, or enhanced physical media form or public or published media, or other by offering the means to create and manipulate information containers with dynamic registers.

The system of the present invention comprises an input device, an output device, a processor, a memory unit, a data storage device, and a means of communicating with other computers, network of computers, or digital-based, supported or enhanced physical media forms or public or published media. These components are preferably coupled by a bus and configured for multi-media presentation, but may also be distributed throughout a network according to the requirements of highest and best use.

The memory unit advantageously includes an information container made interactive with dynamic registers, a container editor, a search interface, a search engine, a search engine editor, system-wide hierarchical container gateways interacting with dynamic container registers, a gateway editor, a register editor, a data collection means with editor, a data reporting means with editor, an analysis engine with editor, an executing engine with editor, databases, and a means of communicating with other computers as above. These components may reside in a distributed fashion in any configuration on multiple computer systems or networks.

The present invention advantageously provides a container editor for creating containers, containerizing storing information in containers and defining and altering container registers. A container is an interactive nestable logical domain configurable as both subset and superset, including a minimum set of attributes coded into dynamic interactive evolving registers, containing any information component, digital code, file, search string, set, database, network, event or process, and maintaining a unique network-wide lifelong identity.

The container editor allows the authoring user to create containers and encapsulate any information component in a container with registers, establishing a unique network lifelong identity, characteristics, and parameters and rules of interaction. The authoring user defines and sets the register with a starting counter and/or mathematical description by utilizing menus and simple graphing tools or other tools appropriate to that particular register. The registers determine the interaction of that container with other containers, system components, system gateways, events and processes on the computer network.

Containers and registers, upon creation, may be universal or class-specific. The editor provides the means to create system-defined registers as well as the means to create other registers. The editor enables the register values to be set by the user or by the system, in which case the register value may be fixed or alterable by the user upon creation. Register values are

evolving or non-evolving for the duration of the life of the container on the system. Evolving registers may change through time, space, interaction, system history and other means.

System-defined registers comprise: (1) an historical container register, logging the history of the interaction of that container with other containers, events and processes on the network, (2) an historical system register, logging the history of pertinent critical and processes on the network, (3) a point register accumulating points based upon a hierarchically rated history of usage, (4) an identity register maintaining a unique network wide identification and access location for a given container, (5) a brokerage register maintaining a record of ownership percentage and economic values, and others.

The present invention also includes user-defined registers. User defined registers may be created wholly by the user and assigned a starting value, or simply assigned value by the user when that register is pre-existent in the system or acquired from another user, and then appended to any information container, or detached from any container.

Exemplary user-defined registers comprise (1) a report register, setting trigger levels for report sequences, content determination and delivery target, (2) a triple time register, consisting of a range, map, graph, list, curve or other representation designating time relevance, actively, assigning the time characteristics by which that container will act upon another container or process, passively, assigning the time characteristics by which that container be acted upon by another container or process, and neutrally, assigning the time characteristics by which that container will interact with another container or process, (3) a triple space register, consisting of a range, map, graph, list, curve or other representation designating the domain and determinants of space relevance, actively, assigning the space characteristics by which that content will act upon another container or process, passively, assigning the space, characteristics by which that content will be acted upon by another container or process, and neutrally, assigning the space characteristics by which that container will interact with another container or process, (4) a domain of influence register, determining the set, class and range of containers upon which that container will act, (5) a domain of receptivity register, determining the set, class and range of containers allowed to act upon that container, (6) a domain of neutrality register, determining the set, class and range of containers with which that container will interact, (7) a domain of containment register, determining the set, class and range of containers which that container may logically encompass, (8) a domain of inclusion register, determining the set, class and

range of containers by which that container might be encompassed, (9) an ownership register, recording the original ownership of that containers, (10) a proportionate ownership register, determining the proportionate ownership of that containers, (11) a creator profile register, describing the creator or creators of that container, (12) an ownership address register, maintaining the address of the creator or creators of that container, (13) a value register, assigning a monetary or credit value to that container, and (14) other registers created by users or the system.

Containers are nestable and configurable as both subset and superset and may be designated hierarchically according to inclusive range, such as image component, image, image file, image collection, image database, or if text, text fragment, sentence, paragraph, page, document, document collection, document, database, document library, or any arrangement wherein containers are defined as increasingly inclusive sets of sets of digital components.

The present invention also includes, structurally integrated into each container, or strategically placed within a network at container transit points, unique gateways, nestable in a hierarchical or set and class network scheme. Gateways gather and store container register information according to system-defined, system-generated, or user determined rules as containers exit and enter one another, governing how containers system processes or system components interact within the domain of that container, or after exiting and entering that container, and governing how containers, system components and system processes interact with that unique gateway, including how data collection and reporting is managed at that gateway. The gateways record the register information of internally nested sub and superset containers, transient containers and search templates, including the grade of access requested, and, acting as an agent of an analysis engine and execution engine, govern the traffic and interaction of those containers and searches with the information resource of which they are the gateway and other gateways. The gateways' record of internally nested and transient container registers, and its own interaction with those containers, is made available, according to a rules-based determination, to the process of the analysis engine by the data collection and/or data reporting means.

The present invention also includes a means of data storage at any given gateway.

The present invention also includes a data collection means, residing anywhere on the network, or located at one or more hierarchical levels of nestable container gateways for

gathering information from other gateways and analysis engines according to system, system-generated or user determined rules. The data collection means manages the gathering of data regarding network-wide user choices, usage and information about information, by collecting it from container and gateway registers as those containers and gateways pass through one another. Such statistics as frequency, pattern, and range of time, space and logical class is collected as directed by the analysis engine, and made that data available to the analysis engine by advancing it directly to the analysis engine, or incrementally, to the next greater hierarchically inclusive collection level. The rules of data collection may be manually set or altered by the system manager, or set by the system and altered by the system in its evolutionary capacity.

The present invention also includes a data reporting means, located at one or more hierarchical levels of nestable container gateways for submitting information to other gateways and analysis engines according to system, system-generated or user determined rules. The data reporting means manages the sending of data from the registers, gateways and search templates in a frequency, pattern, and range of time, space and logical class as directed by the analysis engine, and makes that data available to the analysis engine by advancing it directly to the analysis engine, or incrementally to the next greater hierarchically inclusive reporting level. The rules of data collection may be manually set or altered by the system manager, or set by the system and altered by the system in its evolutionary capacity. The data reporting means may be established to work in concert, in redundancy, or in contiguous or interwoven threads of hierarchically nested containers.

The present invention also includes an analysis engine that receives, reports and collects information regarding the interaction of user searches with gateways and container registers, as well as container registers with other container registers, and container registers with gateways. The analysis engine analyzes the information submitted by the gateways and instructs the execution engine to create new information containers, content assemblages, storage schemes, access routes, search templates, and gateway instructions. The analysis engine includes an editor that provides a system manager with a means of editing the operating principles of that engine, governing data reporting, data collection, search template loading, gateway instructions, and other.

The present invention also includes an execution engine, fulfilling the instructions of the analysis engine, to create new information containers, content sun and superset assemblages, storage schemes, access routes, search templates, and gateway instructions. The execution engine includes an editor that provides a system manager with a means of editing the operating principles of that engine, governing data reporting, data collection, search template loading, gateway instructions, and other.

The present invention also includes a search interface or browser. The search interface provides a means for a searching user to submit, record and access search streams or phrases generated historically by himself, other users, or the system. Search streams or phrases of other users are those that have been historically determined by the system to have the highest probability of utility to the searching user. Search streams or phrases generated by the system are those that have been constructed by the system through the analysis engine based upon the same criteria.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a first and preferred embodiment of a system constructed according to the present invention.

FIG. **2 A** is block diagram of a preferred embodiment of the memory unit.

FIG. **2 B** is an exemplary embodiment of a computer network showing computer servers, personal computers, workstations, Internet, Wide Area Networks, Intranets in relationship with containers and gateways.

FIG. **2B1** is an exemplary embodiment of a computer network showing computer servers, personal computers, workstations, Internet, Wide Area Networks, Intranets in relationship with containers and gateways and exemplary locations of gateway storage in proximity to one or more of the various sites.

FIGS. **2C** through **2H** are exemplary embodiments in block diagram form of computer network components showing a possible placement of nested containers, computer servers, gateways, and the software components named in Fig. 2 A on a network.

FIG. **3A** is a graphical representation for one embodiment of a container having a plurality of containers nested within that container.

FIG. **3C** is a drawing showing elements that might be logically encapsulated by a container. FIG. **4** is a drawing of an information container showing a gateway and registers logically

encapsulating containerized elements.

FIG. **5** is a flowchart showing a preferred method for the containerization process and container editor operating on the communication device.

FIG. **6** is a flowchart showing a preferred method for searching for containers within a node.

FIG. **7** is a flowchart further showing a preferred method for searching for containers over one or more gateways.

FIG. **8** is a flowchart showing a method for performing the data collection and reporting on containers.

FIG. **9** is a flowchart showing the operation of the analysis engine.

FIG. **10** is a flowchart showing the operation of the execution engine.

FIG. **11** is a flowchart showing the operation of the gateway editor.

FIG. **12** is a flowchart showing the operation of the gateway process.

FIG. **13A** is a drawing showing an example of nested containers, gateways, registers, analysis engines and an execution engine prior to container reconstruction as depicted in 13 B, 13 C and 13 D.

FIG. **13B** is a drawing showing the reconstructed nested containers of Figure 13A.

FIG. **13C** is a drawing showing further reconstruction of nested containers, with a container relocated to reside within another container.

FIG. **13D** is a drawing showing a flowchart of the reconstruction process

FIG. **14** is a drawing showing the screen interface of the container editor.

FIG. **15** is a drawing showing the screen interface of the gateway editor.

FIG. **16** is a drawing showing the screen interface of the search interface.

FIG. **17** is a drawing of a generic application program showing a drop-down menu link, and a button link to the containerization process or container editor.

DESCRIPTION OF THE PREFERRED EMBODIMENT

THE SYSTEM

Referring now to FIG. **1**, a preferred embodiment of a system **10** for creating and manipulating information containers with dynamic interactive registers in a computer, media, or publishing network **201** in order to manufacture information on, upgrade the utility of, and develop intelligence in that network **201**, is shown. The system **10** preferably comprises an input device **24**, an output device **16**, a processor **18**, a memory unit **22**, a data storage device **20**, and a communication device **26** operating on a network **201**. The input device **24**, an output device **16**, a processor **18**, a memory unit **22**, a data storage device **20**, are preferably coupled together by a bus **12** in a von Neumann architecture. Those skilled in the art will realize that these components **24**, **16**, **18**, **22**, **20**, and **26** may be coupled together according to various other computer architectures including any physical distribution of components linked together by the communication device **26** without departing from the spirit or scope of the present invention, and may be infinitely nested or chained, both as computer systems within a network **202**, and as networks within networks **201**.

The output device **16** preferably comprises a computer monitor for displaying high-resolution graphics and speakers for outputting high fidelity audio signals. The output device **16** is used to display various user interfaces **110**, **125**, **210**, **300**, **510**, **610**, **710**, as will be described below, for searching for and containerizing information, and editing the container gateways, containers, container registers, the data reporting means and the data collection means, and the search, analysis and execution engines. The author uses the input device **24** to manipulate icons, text, charts or graphs, or to select objects or text, in the process of packaging, searching or editing in a conventional manner such as in the Macintosh of Windows operating systems.

The processor **18** preferably executes programmed instruction steps, generates commands, stores data and analyzes data configurations according to programmed instruction steps that are stored in the memory unit **22** and in the data storage device **20**. The processor **22** is preferably a microprocessor such as the Motorola 680(x)0, the Intel 80(x)86 or Pentium, Pentium II, and successors, or processors made by AMD, or Cyrix CPU of the any class.

The memory unit **22** is preferably a predetermined amount of dynamic random access memory, a read-only memory, or both. The memory unit **22** stores data, operating systems, and programmed instructions steps, and manages the operations of all hardware and software components in the system **10** and on the network **201**, utilizing the communication device **26**

whenever necessary or expeditious to link multiple computer systems **202** within the network **201.**

The data storage device **20** is preferably a disk storage device for storing data and programmed instruction steps. In the exemplary embodiment, the data storage device **20** is a hard disk drive. Historical recordings of network usage are stored on distributed and centralized data storage devices **20**.

The preferred embodiment of the input device **24** comprises a keyboard, microphone, and mouse type controller. Data and commands to the system **10** are input through the input device **24**.

The present invention also includes a communication device **26**. The communication device **26** underlies and sustains the operations of, referring now also to Fig **2** the analysis **400** and execution **500** engines, the data reporting **600** and collection **700** means, the container editor **110,** the search interface **300,** and the search engine **320,** providing the means to search, access, move, copy, utilize or otherwise perform operations with and on data. The communication device **26** utilizes one or more of the following technologies: modem, infrared, microwave, laser, photons, electrons, wave phenomena, cellular carrier, satellite, laser, router hub, direct cabling, physical transport, radio, broadcast or cable TV or other to communicate with other computers, digital-supported television, computer networks, or digital-based or supported public or published media, or physical media forms, on any a local, wide area, public, or any computer-based computer supported, or computer interfaced network, including but not limited to the Internet. It also allows for the functioning and distribution of any container **100** or container component herein described to reside anywhere on any computer system in any configuration on that local, wide area, public, or corporate computer-based or computer related network, or digital-based or supported media form.

Referring now to Figure **2 A**, a preferred embodiment of the memory unit **22** is shown. The memory unit includes: an interactive information container **100**, a container editor **110**, container registers **120**, a container register editor **125**, system-wide hierarchical container gateways **200**, gateway storage **205**, gateway editors **210**, engine editors **510**, a search interface **300**, search engine **320**, analysis engine **400**, execution engine **500**, a data reporting module, **600**, a data reporting editor **610**, a data collection module **700**, a data collection editor **710**, screen interfaces (GUI's) **936**, menu or access buttons from generic computer programs **937**,

and databases **900**, all residing in memory optimized between a data storage means **20** such as magnetic, optical, laser, or other fixed storage, and a memory means **22** such as RAM. The memory unit **22** functions by operating on communications network **12** with a communication device **26** on multiple computer systems **202** within the network **201**. These components will be described first briefly in the following paragraphs, then in more detail with reference to Figures 3 A through 17.

Those skilled in the art will realize that these components might also be stored in contiguous blocks of memory, and that software components or portions thereof may reside in the memory unit **22** or the data storage means **20.**

The present invention includes information containers **100** as noted above. The information container **100** is a logically defined data enclosure which encapsulates any element or digital segment (text, graphic, photograph, audio, video, or other), or set of digital segments, or referring now to FIG. **3 C,** any system component or process, or other containers or sets of containers. A container **100** at minimum includes in its construction a logically encapsulated portion of cyberspace, a register and a gateway. A container **100** at minimum encapsulates a single digital bit, a single natural number or the logical description of another container, and at maximum all defined cyberspace, existing, growing and to be discovered, including but not limited to all containers, defined and to be defined in cyberspace. A container **100** contains the code to enable it to interact with the components enumerated in 2 A, and to reconstruct itself internally and manage itself on the network **201.**

The container **100** also includes container registers **120**. Container registers **120** are interactive dynamic values appended to the logical enclosure of an information container **100**, and serve to govern the interaction of that container **100** with other containers **100**, container gateways **200** and the system **10**, and to record the historical interaction of that container **100** on the system **10**. Container registers **120** may be values alone or contain code to establish certain parameters in interaction with other containers **100** or gateways **200.**

The present invention also includes container gateways **200.** Container gateways **200** are logically defined gateways residing both on containers **100** and independently in the system **10**. Gateways **200** govern the interactions of containers **100** within their domain, and alter the registers **120** of transiting containers **100** upon ingress and egress.

The present invention also includes container gateway storage **205** to hold the data collected from registers **120** of transient containers **100** in order to make it available to the data collection means **700** and the data reporting means **600**, and to store the rules governing the operations of its particular gateway **200**, governing transiting containers upon ingress and egress, and governing the interactive behavior of containers **100** within the container **100** to which that gateway **200** is attached. Gateway storage **205** may be located on gateways **200** themselves, containers **100** or anywhere on the network **202**, **201**, including but not limited to Internet, Intranet, LAN, WAN, according to best analysis and use.

The memory unit **22** also includes an execution engine **500** to perform the functions on the system **10** as directed by the analysis engine after its analysis of data from the data reporting means **600**, the data collection means **700**, and the search interface **300**.

The memory unit **22** also includes a search interface **300**, by which the user enters, selects or edits search phrases or digital strings to be used by the search engine **320** to locate containers **100**.

The memory unit **22** also includes an analysis engine **400** which performs rules based or other analysis upon the data collected from the search interface **300** and the data collection **700** and data reporting **600** means.

The memory unit **22** also includes a data reporting means **600**, by which means the information collected by gateways **200** from transient containers **100** is sent to the analysis engine **400**.

The memory unit **22** also includes a data collection means **700**, by which means the analysis engine **400** gathers the information collected by gateways **200** from transient containers **100**.

The memory unit **22** also includes a container editor **110** for creating, selecting, acquiring, modifying and appending registers **120** and gateways **200** to containers **100**, for creating, selecting, acquiring, and modifying containers, and for selecting content **01** to encapsulate.

The memory unit **22** also includes a register editor **125**, for creating, selecting, acquiring and modifying container registers **120** and establishing and adjusting the values therein.

The memory unit **22** also includes a gateway editor **210**, by which means the user determines the rules governing the interaction of a given gateway **210** with the registers **120** of

transient containers **100,** governing transiting containers upon ingress and egress, and governing the interactive behavior of containers within the container to which that gateway is attached.

The memory unit **22** also includes databases **900,** by which means the analysis engine **400,** the execution engine **500**, the gateways **100**, the editors **110**, **125**, **210**, **510**, **610**, **710**, and the search interface **300**, store information for later use.

The memory unit **22** present invention also includes a search engine **320** by which means the user is able to locate containers **100** and, referring now to Fig. **4**, containerized elements **01**.

The memory unit **22** present invention also includes an engine editor **510,** by which means the user establishes the rules and operating procedures for the analysis engine **400** and the execution engine **500**.

The memory unit **22** present invention also includes a reporting means editor **610**, by which means the user establishes the rules and schedule under which the information collected by gateways **200** from transient containers **100** will be sent to the analysis engine **400**.

The memory unit **22** present invention also includes a collection means editor **710,** by which means the user establishes the rules and schedule under which the analysis engine **400** will gathers the information collected by gateways **200** from transient containers **100**.

The memory unit **22** present invention also includes screen interfaces (GUI's) **936**, specifically designed to simplify and enhance the operations of the container editor **110**, the gateway editor **210**, and the search interface **300**.

The present invention also includes a menu or button access **937**, by which a user utilizing any generic computer program may access the system **10** or the container editor **110** from a menu selection(s) or button(s) within that program.

The present invention also includes a computer, media or publishing network **201**, comprising computers, digital devices and digital media **202** and a communication device **26**, within which the components enumerated in Fig. **2 A** interact, compiling, analyzing, and altering containers **100** and the network **201**according to information gathered from container registers **120**.

The memory unit **22** also includes one or more computers **202**, by which means the components of Fig **1** sustain the operations described in Fig. **2 A.**

The memory unit **22** also includes flat or relational databases **900**, used where, and as required. Databases are used to store search phrases, search templates, system history for the

analysis engine and execution engine, container levels and container, sites and digital elements, or any and all storage required to operate the system.

Referring now to FIG. **2 B,** a drawing of a computer network **201** as a system **10,** showing a possible placement of nested containers **100**, computer servers, gateways **200**, on the sites described below. (Note: Fig. 2 B utilizes in parts the same numbering scheme as Fig. 13 A, 13 B, 13 C, 13 D and as Fig. 2 A.) In FIG. **2 B** various exemplary sites are shown, any or all of which might interact dynamically within the system. Site 1 shows a single workstation with a container and gateway connected to an Intranet. .(Individual containers may be a floppy or CD-Rom to be downloaded or inserted.) Site 2 shows a server with a gateway in relationship to various containers.. Site 3 shows an Internet web page with a container residing on it. Site 4 shows a personal computer with containers and a gateway connected to the Internet. Site 5 shows a configuration of multiple servers and containers on a Wide Area Network.. Site 6 shows a workstations with a gateway and containers within a container connected to a Wide Area Network. Site 7 shows an independent gateway, capable of acting as a data collection and data reporting site as it gathers data from the registers of transiting containers, and as an agent of the execution engine as it alters the registers of transient containers. A container **100** contains the code to enable it to interact with the components enumerated in **2A,** and to reconstruct itself internally and manage itself on the network **201.** The code resides in and with the container in its registers and gateway definitions and controls. Additional system code resides in all sites to manage the individual and collective operation and oversight of the components enumerated in **2A**, with the specific components distributed amongst the sites according to the requirements of optimization.

Referring now to Fig. **2 B 1** various exemplary sites are shown as described above in Fig. **2 B**, with the addition of possible location of one or more gateway storage **205** locations.

Referring now to Figures **2 C** through **2 H**, various exemplary sites with one or more of the logical components of the system 10 in relationship are shown. Site 1 comprises an interactive information container **100**, a container editor **110**, container registers **120**, a container register editor **125**, system-wide hierarchical container gateways **200**, gateway storage **205**, gateway editors **210**, engine editors **510**, a search interface **300**, search engine **320**, analysis engine **400**, execution engine **500**, a data reporting means **600**, a data reporting means editor **610**, a data collection means **700**, a data collection means editor **710**, and databases **900**, all

residing on data storage means **20**, utilizing the memory unit to function **22**, operating on communications network **12** with a communication device **26.**

Site 2 comprises an interactive information container **100**, a container editor **110**, container registers **120**, a container register editor **125**, system-wide hierarchical container gateways **200**, gateway storage **205**, gateway editors **210**, engine editors **510,** search engine **320**, analysis engine **400**, execution engine **500**, a data reporting means **600**, a data reporting means editor **610**, a data collection means **700**, a data collection means editor **710**, and databases **900**, all residing on data storage means **20,** utilizing the memory unit to function **22**, operating on communications network **12** with a communication device **26.**

Site 3 comprises an interactive information container 100, a container editor **110**, container registers **120**, a container register editor **125**, hierarchical container gateways 200, gateway storage 205, gateway editors **210**, and databases **900**, all residing on data storage means **20,** utilizing the memory unit to function **22**, operating on communications network **12** with a communication device **26.**

Site 4 comprises an interactive information container **100**, a container editor **110**, container registers **120**, a container register editor **125**, hierarchical container gateways **200**, gateway storage **205**, gateway editors **210**, a search interface **300**, and databases **900**, all residing on data storage means **20**, utilizing the memory unit to function **22**, operating on communications network **12** with a communication device **26.**

Site 5 comprises an interactive information container **100**, container registers **120**, a container register editor **125**, hierarchical container gateways **200**, gateway storage **205,** and databases **900**, all residing on data storage means **20**, accessed and utilized by non-resident memory unit **22**, operating on communications network **12** with a communication device **26.**

Site 6 includes an independent analysis engine **400,** execution engine **500**, data collection means **700**, and data reporting means **600** gateway editors **210**, engine editors **510,** a data reporting means editor **610**, a data collection means **700**, a data collection means editor **710**, and databases **900**, all residing on data storage means **20**, utilizing the memory unit to function **22**, operating on communications network **12** with a communication device **26.**

Referring now to FIG. **3 A** and FIG. **3 B**, a block diagram of several nested information containers is shown, including examples of elements, e.g., code **1100**, text **1200**, audio **1300**, video **1400**, photograph **1500**, graphic images **1600**, and examples of possible container level

classifications in increasing size, e.g., element **10900000**, document **10800000**, database **10700000**, warehouse **10600000**, domain **10500000**,and continuing increasingly larger on Fig **3 (B)**, subject **10400000**, field **10300000**, master field **10200000**, species **10100000**. Containers may be infinitely nested and assigned any class, super class or sub class scheme and description by the creator of the container to govern nesting within that container. In addition to digital elements, containers may also include system process and components, including containerization itself.

Referring now to FIG. **3 C**, a block diagram of an information container system is shown, listing, without any relationship indicated, some of the possible system components and processes, or sets thereof, that may be encapsulated as elements **01** in an information container **100.** An information container 100 may include one or more of the following: any unique, container **100**, gateway **200**, output device **16**, input device **24**, output device process **160**, input device process **240**, data storage device **20**, data storage device process **2000**, processor **18**, bus **12**, content **01**, search process **02**, interface **04**, memory unit **22**, communication device **26**, search interface **300**, search process **98**, network **201**, class of device, process or content **999**, class of process at any unique class of device **990**, process at any unique device **99**, editor **110**, **125**, **210**, **510**, **610**, **710**, engine **320**, **400**, **500**, containerization process **1098**, or process **08**.

Any container may include (n) other containers, to infinity. The use of value evolving container registers **120** in conjunction with gateways **200**, data reporting modules **600**, data collection modules **700**, the analysis engine **400**, and the execution engine **500** provides the information container **100** with extensive knowledge of the use, operation of its internal contents, prior to, during and after those contents' residence within that container **100**, and extensive knowledge of the use, operation and contents of the system **10** external to itself, and allows the container **100** to establish and evolve its own identity and course of interaction on the system **10**. Further, containers **100**, as logical enclosures, can exist and operate independent of their digital contents, whether encapsulating audio, video, text, graphic, or other.

Referring now to FIG. **4**, a block diagram of an information container **100** is shown. The information container **100** is a logically defined data enclosure which encapsulates any element, digital segment (text, graphic, photograph, audio, video, or other), set of digital segments as described above with reference to FIG. **3 (C)**, any system component or process, or other

containers or sets of containers. The container **100** comprises the containerized elements **01**, registers **120** and a gateway **200**.

Registers **120** appended to an information container **110** are unique in that they operate independently of the encapsulated contents, providing rules of interaction, history of interaction, identity and interactive life to that container **100** through the duration of its existence on a network **201**, without requiring reference to, or interaction with, its specific contents. They enable a container **100** to establish an identity independent of its contents. Additionally, registers **120** are unique in that their internal values evolve through interaction with other containers **100**, gateways **200**, the analysis engine **400**, the execution engine **500**, and the choices made by the users in the search interface **300**, the container editor **110**, the register editor **125**, the gateway editor **210**, the engine editor **510**. Registers **120** are also unique in that they can interact with any register of a similar definition on any container **100** residing on the network **201**, independent of that container's contents. Registers **120**, once constructed, may be copied and appended to other containers **100** with their internal values reset, to form new containers. Register values, when collected at gateways **200** and made available to the analysis engine **400** through the data collection means **700** and the data reporting means **600**, provide an entirely new layer of network observation and analysis and operational control through the execution engine **500**. Registers **120** accomplish not only a real time information about information system, but also a real time information about information usage on a network. Further, because the user base of a network determines usage, the system **10**, in gathering information about information usage, is observing the choices of the human mind. When these choices are submitted to the analysis of a rules-based or other analysis engine **400**, the system **10** becomes capable of becoming progressively more responsive to the need of the user base, in effect, learning to become more useful by utilizing the execution engine **500** to create system-wide changes by altering the rules of gateway **200** interaction and thereby altering the registers **120** of transient containers **100** and establishing a complete evolutionary cycle of enhanced utility.

Further, in establishing the pre-defined registers as described in the following four paragraphs, the following unique aspects of information about information are utilized for the first time: 1) the dynamic governance of information according to its utility through time, in active, passive and neutral aspects, as explained below; 2) the dynamic governance of

information according to its utility through space in active, passive and neutral aspects, as explained below; 3) the dynamic governance of information according to its ownership, as explained below; 4) the dynamic governance of information according to its unique history of interaction as an identity on a network, as explained below; 5) the dynamic governance of information according to the history of the system on which it exists, as explained below; 6) the dynamic governance of information according to established rules of interaction, in active, passive and neutral aspects, as explained below; 7) the dynamic governance of information according to the profile of its creator, as explained below; 8) the dynamic governance of information according to the value established by its ongoing usage, as explained below; 9) the dynamic governance of information according to it distributed ownership, as explained below; 10) the dynamic governance of information according to what class of information it might be incorporated into, and according to what class of information container it might incorporate, as explained below; 11) the dynamic governance of information according to self-reporting, as explained below.

Referring now to Fig **4**, registers **120** may be (1) pre-defined, (2) created by the user or acquired by the user, or (3) system-defined or system-created. Pre-defined registers **120** are those immediately available for selection by the user within a given container editor as part of that container editor, in order that the user may append any of those registers **120** to a container **100** and define values for those registers **120** where required. Registers **120** created by the user are those conceived and created by a specific user or user group and made immediately available for selection by the user or user group in conjunction with any of a wide number of container editors, in order that the user may append any of those registers **120** to a container **100** and define values for those registers **120** where required. Registers **120** acquired by the user are those registers existing network-wide **201**, created by the user base, that might be located and acquired by the user in order that the user may append any of those registers **120** to a container **100** and define values for those registers **120** where required. System-defined registers are those registers whose values are set and/or controlled by the system **10**. System-created registers are those registers created by the system **10**.

Registers **120** are user or user-base created or system-created values or ranges made available by the system **10** to attach to a unique container, and hold system-set, user-set, or system-evolved values. Values may be numeric, may describe domains of time or space, or may

provide information about the container **100**, the user, or the system **10**. Registers **120** may be active, passive or interactive and may evolve with system use. Pre-defined registers include, but are not limited to, system history **110000**, container history **101000**, active time **102000**, passive time **103000**, neutral time **104000**, active space **111000**, passive space **112000**, neutral space **113000**, containment **105000**, inclusion **106000**, identity **114000**, value **115000**, ownership **107000**, ownership addresses **116000**, proportionate ownership **117000**, creator profile **108000**, receptivity **118000**, influence **119000**, points **109000**, others **120000**, reporting **121000**, neutrality **122000**, acquire **123000**, create **124000**, content title **125000**, content key phrase(s) **126000**, and content description **127000**, security **12800**, and parent rules **129000**.

Pre-defined registers comprise an historical container register **101000**, logging the history of the interaction of that container **100** with other containers, events and processes on the network **201**, an historical system register **110000**, logging the history of pertinent critical and processes on the network, a point register **109000** accumulating points based upon a hierarchically rated history of usage, an identity register **114000** maintaining a unique network wide identification and access location for a given container specifying a unique time and place of origin and original residence, a proportionate ownership register **117000** maintaining a record of ownership percentage and economic values, and others **120000**.

User-defined registers include a report register **121000** setting trigger levels for report sequences, content determination and delivery target, three time registers, consisting of a range, map, graph, list, curve or other designating time relevance, **102000** assigning the time characteristics by which that container will act upon another container or process, **103000** assigning the time characteristics by which that container be acted upon by another container or process, and **104000** assigning the time characteristics by which that container will interact with another container or process, three space registers, consisting of a range, map, graph, list, curve or other designating the domain and determinants of space relevance, **111000** assigning the space characteristics by which that content will act upon another container or process, **112000** assigning the space, characteristics by which that content will be acted upon by another container or process, and **113000** assigning the space characteristics by which that container will interact with another container or process, a domain of influence register **119000**, determining the set, class and range of containers upon which that container will act, a domain of receptivity register **118000**, determining the set, class and range of containers allowed to act upon that

container, a domain of neutrality register **122000**, determining the set, class and range of containers with which that container will interact, a domain of containment register **105000**, determining the set, class and range of containers which that container may logically encompass, a domain of inclusion **106000** register, determining the set, class and range of containers by which that container might be encapsulated, an ownership register **107000**, recording the original ownership of that containers, a creator profile register **108000**, describing the creator or creators of that container, an ownership address register **116000**, maintaining the address of the creator or creators of that container, a value register **115000**, assigning a monetary or credit value to that container, other registers **120000** created by users or the system, a reporting register **121000**, determining the content, scheduling and recipients of information about that container, a neutrality register **122000**, an acquire register **123000**, enabling the user to search and utilize other registers residing on the network, a create register **124000**, enabling the user to construct a new register, a content title register **125000**, naming the contents of the container, a content key register, **126000**, identifying the container contents with a key phrase generated by the user and/or the system based upon successful usage of that phrase in conjunction with the utilization of the information within that container **100**, a content description register **127000**, identifying the container contents with additional description, a security register **128000**, controlling container security, and a parent container register **129000**, storing the rules governing container interaction as dictated by the parent (encapsulating) container.

The container also includes a gateway **200** and gateway storage **205.**

Gateways **200** are logically defined passageways residing both on containers **100** and independently in the system **10**. Gateways **200** govern the interactions of containers **100** encapsulated within their domain by reading and storing register **120** information of containers entering and exiting that container **100**.

The present invention also includes container gateway storage **205.** Gateway storage **205** stores information regarding the residence, absence, transience, and alteration of encapsulated and encapsulating containers **100**, and their attached registers **120, holding** the data collected from registers **120** of transient containers **100** in order to make it available to the data collection means **700** and the data reporting means **600**, and storing the rules governing the operations of its particular gateway **200**.

Referring now to FIG. **5**, a flow chart of the preferred method for creating a container **100**.is shown.

Input is received from the user selecting a container level through use of a drop-down menu **10100**. A menu of all possible container classes within the subset and superset scheme of multiple hierarchically nested containers, i.e.; element, document, file, database, warehouse, domain, and more, is displayed on the output device **10200.** Input is received from the user selecting a class **10300.**

A graphic representation of a container in that class, with registers common to all containers as well as registers unique to its class is displayed **10301.**

Input is received from the user choosing to "create" **10400**, "edit" **10500**, or "locate" **10600**.

When the input of "create" **10400** is received from the user, a container template in that class appears **10410**. Input from the user is then received adding or selecting a register 10540 to append to that container template. When input is received from the user adding a register, a list of registers that might be added to that class of container is made available to select **10550**. Input is received from the user selecting a register **10560** and editing it **10570**. The menu returns to "add or select" **10540**.

If the input of "locate" **10600** is received from the user, the system prompts the user to enter the identity of the container or class of containers **10605**. The system locates the container(s) **10610**. Input is received from the user selecting a container **10620**. The system prompts the user for a security code for permission to access the container for template use, or to alter its registers, or to alter its content **10630**. . Input is received from the user entering a name and password providing access to one of the security levels **10640.** Input is received from the user editing the container accordingly by transition to step **10500** and performing the steps for editing.

If the input of "edit" **10500** is received, a list of containers available to edit at that level is shown **10510**. Input is received from the user selecting a container **10520**. That container appears, available to edit **10530**. Input is received from the user selecting "add" or "select" registers **10540** by the user clicking on the graphically depicted register, or from a drop down menu. Input is received from the user selecting the register to edit **10560**. Input is received from the user selecting "modify" or "delete" for that register **10565**. If input is received from the user

to "delete," that register is severed from the container. If input is received from the user to "modify", the register editor **10570** screen appropriate to that register appears, i.e., an x-y type graph to define a curve of relevant active time, in which the user manipulates the x-y termini, scale and curve, or a global map in which Input is received from the user selecting the locale of active space, whether zip code, city, county, state, country, continent, plant or other. When input is received from the user saving the definition, the screen returns to the main container screen to make another selection available. . Input is received from the user defining as many registers as he chooses. One of the registers may be named "new register." Input is received from the user selecting the new register, and if chosen by the user, defining a wholly unique and new kind of register by the user entering input into the register editor **125.**

When the input is received from the user choosing to add a register, a list of registers that might be added to that class of container are made available to select **10550**. Input is received from the user selecting a register **10560** and editing it **10570**. The menu returns to "add or select" **10540**, and in turn to Input – Select Container.

Input may then be received from the user choosing to add, modify, or delete the container contents **10700**. Once the registers are defined, input is received from the user indicating completion and the interface reverts to the container editor. When input is received from the user choosing "select component" (to select the component to containerize) from the main menu bar **10700, a** window appears allowing the user to select any file, component, or other container. If for example, the user were creating a warehouse container, and wishes to incorporate several databases into that container, input would then be received from the user selecting "database." The program would prompt the user for the location (directory) of that database or container. If the requested selection is not containerized, input may then be received from the user choosing to containerize the element at that time, after which the program returns to "select component." Once input is received from the user defining the database location, the program logically encases the directory or directories in the defined container. The above procedure may be repeated as many times as desired to include multiple databases within a single container. While logical simplicity would dictate that all containers within a container be of the same subset, it would be possible for input to be received from the user choosing containers of any subset to include in the container. When input is received from the user choosing "finished," the container is created with a unique network identity, preferably through

some combination of exact time and digital device serial number, or centralized numbering system, or other means. The container **100** contains all digital code, including data and program software from the selected items or containers.

Input may then be received from the user to publish the container **11100** at a user-identified or system suggested location **11200** to be selected **11400**.

Input is received from the user to "publish", from the main menu bar **11100**. Input is received from the user choosing to leave the container where it was created, move or copy it to another drive, directory, computer, or network the user designates, or select the location from location options offered by the system **11200**, or submit, or duplicate and submit, the container to the analysis engine **400** for intelligent inclusion in other containers, thus allowing the system to publish the container as instructed or choose the residence of the container **11400**.

If input is received from the user to choosing to "move," or "copy" a browse function allows the user to name the new location or browse a list of possible locations. If input is received from the user choosing to "submit," a browser function allows the user to name the analysis search engine **310** or browse a list of possible analyses engines. When input is received from the user choosing the residence of the container **11300**, the program restores the search interface screen.

Referring now to FIG. **6**, a flow chart of the method for searching for containers **100**.

When input is received from the user selecting "search interface" from the main title bar, the search interface screen appears. The user is given the choice of containerizing selected content or requesting that container levels be displayed **30100**. From a drop down menu another menu appears allowing input to be received from the user selecting the container level **30200**. Input is received from the user selecting the container level (from the smallest component to the whole system) **30300.**

Input is received **30310** from the user selecting the phrases, containers or components, which then are re-submitted to the same process, until the input is received from the user selecting a specific site or container.

The search phrase, whether containerized or not, is submitted simultaneously to the search engine **30400** and the analysis engine **30500**.

The screen then reports in a selection menu, the number of applicable sites found by the search engine **30410**, the number of historically proven applicable sites found by the analysis

engine **30410**, the number of historically proven applicable containers at the selected container level or any container level found by the analysis engine **30410**, and the number of historically proven new search phrases or digital segments found by the analysis engine **30320**. . Input is received from the user selecting one of the named sets above **30330**. If input is received from the user choosing the search engine, the search interface lists the applicable site titles with a brief description **30410**. If input is received from the user choosing the site list of the analysis, the search interface lists the applicable site titles with a brief description **30410**. . If input is received from the user choosing the container list of the analysis engine, the search interface lists the applicable container titles with a brief description **30410**. . If input is received from the user selecting a container **30420**, the system offers the means to view titles and descriptions of sub-containers at any chosen class level. . If input is received from the user choosing the phrase list of the analysis engine, the search interface lists the applicable phrases or digital segments with a brief description **30320**. The search and search result cycle repeats until input is received from the user choosing to go to an individual container or site.

Input is received from the user entering text or any digital string describing his search objectives into a text or search box. When input is received from the user submitting the search string, the system provides the option of containerizing the search through the container editor **110.** Once the search container **101** is created, the system restores the search interface **300** screen the user.

Input is received from the user selecting "search", "supported search" or "both" from another drop-down menu and from submitting the search. When input is received from the user selecting "search" 30310**,** the search phrase is submitted to the search engine **30400**, which searches both content and the appropriate container registers, as pre-indexed in the search engine, and returns a list of appropriate locations, components or containers. When input is received from the selecting "supported search", the search phrase is submitted to the analysis engine search support, which returns a list, in a drop-down menu, of search phrases or individual containers, for any and all container levels, used by other users or created by the system and known to be historically successful for the described effort and the described searching user, as per the results of the analysis search engine. Input is received from the user selecting a new search phrase or specific container from the drop down menu **30330**. When input is received from the user choosing a new search phrase, that phrase is also submitted to the

analysis engine **30500** which returns a list of pre-compiled historically proven sites, components or containers associated with that search phrase **30320**. Input is received from the user choosing a selection **30420** and the system calls up that specific site, container or component. If input is received from the user selecting a specific site, container or component at any time during the search process, that element is called up by the system **30440**.

Input is received from the user choosing to containerize a search or select a container level in which to search **30100**. When input is received from the user choosing to containerize the search, the software moves to the container editor as described in Fig. **5**, and then returns the user to the search interface screen. Input is received from the user selecting to search a specific container level or the whole network. The system shows the available levels **30200**. Input is received from the user selecting a container level **30300**, and entering the text or digital component comprising the search string **30310**. The system searches the containers **30400** while simultaneously submitting the search string to the analysis engine **30500**. While the system is accessing containers, sites or templates **30700**, the analysis engine **30500** inquires of the appropriate database **30600** to access historically successful containers, sites or search templates corresponding to the search request **30700**, which is then shown on another portion or option of the search interface, either as available containers or sites **30410** or as search template options **30320.** On one portion or option of the search interface screen the corresponding containers or sites are listed and/or previewed for selection **30410**. Input is received from the user selecting the container to access **30420**. The system accesses that container **30430** and shows it on the screen **30440** for user review. Input is received from the user selecting an operation, i.e., preview, read, purchase, move, copy, lease, in any composed schedule with operations assigned specific values **30460**, and the system obtains the specified result **30470**. The selection of the operation including any interaction with any uniquely defined container **100** is recorded **30800** by the container gateway (Fig. 2 A, **200**), stored in the gateway storage **205** and made available to the analysis engine (Fig. **9**) by the data collection and reporting means (Fig. **8**). Reporting and collection occurs on a regular basis according to user determined times or rules. The analysis engine compiles and analyzes selections according to various rules-based systems applicable to the particular container area of residence in cyberspace.

Input is received from the user selecting the container or site **30410**, proceeding as described above, or selecting a search template **30330**, and editing it to re-enter the search

30310. All operations on Fig. 6 utilize the communication device 26 whenever necessary or expeditious.

Referring now to FIG. 7, a flow chart of the search process is shown. Steps in FIG. 7 repeated from FIG. 6 are given the same reference number as in FIG. 6 for convenience and ease of understanding. Fig. 7 commences with "SEARCH TRANSITS GATEWAY 32100", continuing from Fig. 6, "SYSTEM SEARCHES CONTAINERS 30400". The submitted search 32100 transits the gateway 200. The gateway 200 interacts with the container registers 32200. The gateways 200 store the information downloaded from the registers 32300, and the container registers are altered 32500. The container registers 120 then interact with the registers 120 of the encapsulated search, which registers, and the values set within, have been constructed and appended to the search through the search interface 32600. Values are exchanged and compared and operations performed under the rules governing both interacting containers 100, and the rules governing the search container 100 and any gateway 200. The search engine 320, operating under the principles and means of search engines presently existing as described elsewhere, **then** provides to the search interface 32600 a list of containers 100 meeting the requirements of the search and its appended registers, as well as additional search options 32900. The gateway 200 reports and makes available for collection to the analysis engine 400 the information obtained from the interaction 32400. On a periodic basis defined by the user or a rules-based system, the analysis engine 400 (Fig. 9) stores in databases 900, analyzes and instructs the execution engine 500, and the execution engine 500 executes changes in the system components as defined below. (Fig. 10). All operations on Fig. 7 utilize the communication device 26 whenever necessary or expeditious.

On the remaining figures, shapes referring to other figures, to operations external to the scope of the present figures, or to the subject of the present drawing, are indicated with dashed lines, and are shown only to place the described operations in the context of continuous and continual operations external to the drawing.

Referring now to FIG. 8, a flow chart of the preferred process for collecting and reporting information on containers is shown. The data reporting 600 and data collection 700 means utilizes subroutines within the analysis engines 400 and gateways 200 to submit and collect register information and sub level analysis to other analysis engines 400 or other gateways 200 of a higher (larger) logical set in a set pattern and frequency defined by the administrator.

Input is received from the user selecting "data reporting" **70100** from the "edit gateway" drop-down menu. Container levels are displayed **70200**. Input is received from the user selecting container level **70300.** A menu of all possible gateways **70320** and analysis engines **70330** residing on gateways on the above defined container class appears, depicted graphically as a tree of analysis engines and gateways at that container level. Input is received from the user selecting "source" from "source or destination." Input is received from the user **70400** selecting a container, containers, or class of container by clicking on the graphically depicted container(s) or container level on a display device. Input is received from the user **70410** selecting "destination" from "source or destination" Input is received from the user **70500** selecting an analysis engine, analysis engines, or class of analysis engine by clicking on the graphically depicted analysis engine(s) or analysis engine level on a display device. A time scheduler is displayed. Input is received from the user **70510** selecting the reporting frequency for the selected gateways to report data to the selected engines. The data from the gateways is thenceforth continuously moved or copied to the analysis engines by the system **10** utilizing the execution engine **500** according to the defined schedule, rules and pattern **70420, 70520**.

Input is received from the user selecting "choose container level" **70300** from the gateway editor drop-down menu. A menu **70320** appears listing the classes of containers on the system within the defined subset and superset scheme of multiple hierarchically nested containers, i.e.; element, document, file, database, warehouse, domain, appears. Input is received from the user selecting the class of containers. A graphic representation of that container level throughout the system appears. Input **70300** is received from the user selecting individual containers or all the containers in that class.

From the gateway editor drop-down menu input **70100** is received from the user selecting "data collecting" A menu of all possible gateways and analysis engines residing on gateways on the above defined container class appears, depicted graphically as a tree of analysis engines, and gateways at that container level. Input **70510** is received from the user selecting "source" from "source or destination." Input is received from the user selecting a container, containers, or class of container by clicking on the graphically depicted container(s) or container level. Input **70510** is received from the user selecting "destination" from "source or destination." Input **70510** is received from the user selecting an analysis engine, analysis engines, or class of analysis engine by clicking on the graphically depicted analysis engine(s) or analysis engine

level. A time scheduler appears. Input **70510** is received from the user selecting the collecting frequency for the selected engines to collect data from the selected gateways. The data from the gateways is thenceforth continuously moved or copied to the analysis engines by the system **10** utilizing the execution engine **500** according to the defined schedule, rules and pattern.

The data collection **700** means, utilizing the communication device **26** and an execution engine **500,** comprises one or more subroutines or agents programmed to travel through the network collecting the accumulated data and analyses from selected analysis engines, gateways or selected subset level of analysis engines or gateways (as above) in a pattern and frequency defined by the gateway administrator at a given container level. Input **70510** is received from the user or administrator, defining the collection and reporting of data, thus controlling permission within his gateway, and being subject to permission levels defined by others beyond his gateway.

Input is received from the user or gateway administrator selecting collection or reporting **70100** and the system shows the container levels available **70200**. Input is received from the user selecting a container level **70300**. Input is received from the user selecting "gateway" **70400** or "engine" **70500**. The system shows gateways **70320** or engines **70330** associated with that level. Input is received from the user editing the reporting parameters associated with a gateway or a class of gateways **70410** or an engine or class of engines **70510**. Input is received from the user selecting the collecting frequency for the chosen engines. When input is received from the user choosing to user save the definition, the screen returns to the main container screen, step **70100** to make another selection available. Input is received from the user choosing to repeat the cycle, choosing "destination" to describe the destination analysis engines and the data collecting frequency from those destination analysis engines. The data collection means **700** collects the accumulated gateway information in a pattern and frequency defined by the gateway administrator or user at a given container level.

The system utilizing the execution engine (see Fig. **10**) distributes the new parameters to the gateways **70420** or engines **70520** by the communication device **26**. Using the new parameters the gateways report to the analysis engines **70430** after, in some cases, conducting sub-analysis **70440,** or using sub-analysis **70440** to submit directly to specified gateways under certain conditions and parameters, and the analysis engines collect from the gateways **70530**.

The analysis engine uploads, downloads and utilizes information to databases **900** to conducts its analysis.

The invention includes an analysis engine **400**. Through the data reporting **600** means and data collection **700** the analysis engine **400** receives data and sub-analysis from the search interface and the gateways. Data includes, for each gateway **200**, the frequency and grade of access, the description of the user accessing, the identity of the container **100** accessing, the register parameters, and the historically accumulated register data.

Referring now to FIG. **9**, a flow chart of the operation of the analysis engine **400** is shown. Analysis engines **400** may reside at any gateway or anywhere in the system 10. The analysis engine **400**, operating under its own programmed sequence, utilizing the communication device **26**, works, by means of programmed rules of logical, mathematical, statistical or other analysis upon gateway and register information, in continuous interaction with the search process **410** and the data collection and reporting process **420** to analyze, determine and compile instructions **40100** on container construction **40110** to containerize in an automated process **40115**, on container contents **40120** to move, copy or delete containers **40125**, on storage schemes **40130** to move or copy containers to new storage **40135**, on access routes **40140** to alter gateway pointers to sought information **40145**, on search templates **40150** to add, delete or change search phrases and the referenced objects indicated by those search phrases **40155** and on gateway instructions **40160** to alter gateway registers and pointers **40165**.

Thus, analyses might include, but are not limited to, the physical locus of the users accessing, the demographic classification of the users accessing, the access frequency for a given container, the range or curve of time relevance affecting a container, the range or region of space relevance affecting a container **100**, the number or number of a specific type of container **100** transiting a gateway **200**, the hierarchically graded usage of containers **100** or container contents **01** compared with the demographic of those users accessing the container, the hierarchically graded usage of containers **100** or container contents **01** compared with search phrases entered into the search interface **300**, the hierarchically graded usage of containers **100** or container contents **01** compared with search phrases entered into the search interface 300 compared with the demographic of the users accessing, the number of pertinent containers nested within a given container **100**. Once an analysis is accomplished, the result is compared to

pre-programmed rules triggering instruction sets (such as moving a container to nest within another container).

Instructions are then sent to the execution engine **40200**, which utilizes the communication device **26** to execute the instructions derived from the analyses. These containerized instructions transit the gateways **40300** and are utilized in the gateway process (Fig. **12**)

Referring now to FIG. **10**, a flow chart of the operation of the execution engine is shown. The execution engine **400**, operating under its own programmed sequence in response to the instructions from the analysis engine **50100**, utilizing the communication device **26**, works in continuous process as its containerized execution instructions transit the gateways **50200** to create containers **50210** in an automated containerization process **50215**, alter container contents **50230** by moving or copying containers to new containers **50235**, to alter storage **50240** by moving or copying containers to new storage **50245**, to alter access routes **50250** by altering gateway pointers **50255**, to alter search templates **50260** by adding, changing and deleting search phrases and the referenced objects indicated by those search phrases **50265**, to alter gateway instructions **50270** by altering gateway registers and pointers **50275**. The execution works in a continuous loop with the gateway process **50300**, the data collection and reporting process **50400** and the analysis engine process **50300**.

The invention includes gateways **200**. Gateways may be placed and reside anywhere on the network where containers transit. Gateways also reside on any or all containers. The gateway reads and stores the chosen register information from transient containers entering or exiting its logical boundaries. The resident analysis search engine, if any, performs the specified level of analysis. Data and analysis is both held for the collection means according to the pattern and timing specified in the data reporting **600** editor and submitted according to the pattern and timing specified in the data collection means editor **700.**

The gateways are network-wide, hierarchical, and nestable, and reside with a container encompassing any component, digital code, file, search string, set, database, network, event or process and maintaining a unique lifelong network wide identity and unique in all the universe historical identity, or may be strategically placed at such container transit points to gather and store register information attached to any such container, according to system-defined, system-generated, or user determined rules residing in its registers defining the behavior of those

containers and components as they exit and enter one another, or interact with one another or any system process or system component within the logical domain of that container, or after exiting and entering that container, or defining how they interact with that unique gateway.

Gateway's registers comprise both system-defined and user-defined registers, alterable by author, duration, location, network-wide history, individual container history and/or interaction with other containers, gateways, networks or media, and evolve according to that gateway's history on a computer network, or according to the network history of events and processes, or according to that information component's interaction with other information containers, components, system components, network events or processes.

Referring now to FIG. **11,** a flow chart of the gateway editor is shown. From the main title bar input is received from the user selecting "containerize" or "gateway level" **20100**. When input is received from the user selecting "containerize" the system enters the container editor process **110.** When input is received from the user selecting "gateway," the system shows the gateway levels available **20200**. A menu of all possible gateways within the subset and superset scheme of defined multiple hierarchically nested gateways appears. Input is received from the user selecting the gateway level **20300.** The system searches the gateways **20500** to locate the available gateway templates **20700** and the available gateways **20600**. Input is received from the user selecting the gateway **20610** or gateway level template **20720**. The system goes to the gateway **20620** or to the template **20720**. A graphic representation of the chosen gateway **20630** or template **20730** appears. Input is received from the user to edit **20640** or create a gateway **20740**. Once completed, input may be received from the user selecting "analysis level" from the gateway **200** drop-down menu, to select the level of analysis in a multi-level analysis sequence to be accomplished at the local level by a gateway-resident analysis engine. The user accesses the container editor to containerize (Fig. **5**). Input is received from the user selecting the registers by clicking on the graphically depicted register, or from a drop down menu.). Input is received from the user setting the registers as described elsewhere in ("container registers"). Input is received from the user selecting or defining the rules governing the interaction of that gateway with transient containers. Input is received from the user selecting or defining the rules governing the interaction of containers existing within the logical domain of the container 100 to which that gateway is attached. The user publishes the gateway (Fig. **5**). Input is received from the user selecting "residence" from the main menu bar.

). Input is received from the user choosing to leave the gateway where it was created, move it to container on another drive, directory, computer, or network. If the user chooses "move," a browse function allows the user to name the new location or browse a list of possible locations. Once input is received from the user choosing the residence of the gateway, the program restores the search interface screen.

The invention includes a data reporting means editor **610**, and a data collection means editor **710,** Fig. 2 A, as a menu option under the gateway editor **210**.

The present invention also includes a gateway process.

Referring now to FIG. **12**, a flow chart of the gateway process is shown. A system operation, search process or element container or process container is shown in transit **21100** passing through a gateway **21200**. The container, operation or process interacts with the gateway **21300**, uploading, downloading and exchanging information with the container, operation or process. The gateway stores container information **21400** and the container registers are altered **21500**. The container registers also interact with the search interface **21600**. The gateways report the register information or make it available for collection by the data reporting and collection means (Fig. **8**) operating on the communication device **26** to provide the information to the analysis engine **21800**, which stores **90100**, analyzes and instructs the execution engine **21900**, which processes and instructions are also stored **90100** by the execution engine upon receipt.

All operations in Fig. **12** utilize the communication device **26** whenever necessary or expeditious.

Referring now to FIG. **13 A,** a drawing of nested containers **100** prior to the container modification process on a network **201** is shown. (Note: The same container numbering scheme is used in Fig. 13 A, 13 B, 13 C, 13 D and in 2 B.) Information containers **505** and **909**, residing within container **908**, operating under the rules governing container interaction within that container **908** downloaded to container **505** and **909** from gateway **9081** upon their entrance to container **908,** which rules had been downloaded from execution engine **500** acting under the direction of analysis engine **400**, and under the rules programmed into their own registers **404120**, **909120**, compare the specified (by those rules) set of registers **404120**, **909120**, i.e., time and space, and determine a container **404** encapsulated within **505** would be more appropriately encapsulated within container **909**.

Referring now to FIG. **13 B** a drawing of nested containers during a container modification process on a network **201** is shown. Container **404** is moved to reside with container **909**. As the container **404** exits container **505**, the gateway of container **505**, being gateway **5051**, operating under the rules governing container interaction with a gateway 5051 upon egress or egress as programmed in the gateway editor **210** and modified by the execution engine **500** executing the instructions of the analysis engine **400**, or any greater logical analysis engine **408** providing execution instructions to an execution engine **508** operating in a larger encompassing container **108** entering through that container's gateway **208** or an independent gateway **707**, or sub-analysis engine operating at any gateway level, records the register information of container **404.** The gateway **5051** reports the transaction to the gateway **9081** of container **908**, being the next higher logical container. Gateway **9081** holds in gateway storage **205** the information until collected by one or more data collection processes **700**, or reported to one or more data reporting processes **600**, serving one or more analysis engines **400** residing independently on the system **10** or an analysis engine at higher logical container **303**. The analysis engine **400,** comparing reports of user hierarchically graded usage under the operations of the search engine **320** and the search interface **300**, on information container **808** after receiving reports from the data reporting means of container **404** being moved to container **909** determines, i.e., that the number of time and space relevant containers residing within container **909** is sufficient to warrant an action, and directs the execution engine **500** to copy container **909**, nested within container **908**, to a third information container **808**. As the copy instruction from execution engine **500** transits the gateway of container **908**, the gateway **9081** records the instruction. The copy instruction interacts with the registers **909120** of container **909** regarding the rules governing its copying to another location. Once approved by the governing rules of registers **909120** appended to container **909**, container **909** is duplicated. As the duplicate container **909** exits the container **908**, the gateway records the register information **909120** of container **909**, and the registers **909120** of container **909** are altered by special instructions from gateway **9081** under the rules residing in gateway **9081** regarding ingress and egress and the rules residing in the registers **909120** of container **909** regarding alteration by gateways upon ingress and egress. Passing through independent gateway **707**, the register information **909120** is recorded, and awaits data collection or reporting 700, **600.** As container **909** enters container **808**, the gateway records the register information **909120** of container **909**, the registers **909120**

of **909** are altered by special instructions from gateway **8081**, operating under the rules as described in the paragraph above, and container **909** takes up residence within container **808.**

Referring now to FIG. **13 C,** a drawing of nested containers after the container modification process on a network **201** process is shown. Container **909**, now also logically residing within container **808,** commences to interact with other containers **606** in **808** under the rules governing container interaction within container **808** as received from gateway **8081** upon transiting that gateway, and under the rules of registers **606120, 909120 of** the interacting containers **606, 909**, operating under the rules as described in the paragraph above. Through data collection and reporting **700, 600**, analysis engine is appraised of container's **909** new duplicate residence. I.e., operating under the registers of space relevance, a body of law pertaining to Boston Municipal tax law may be housed in a container holding Massachusetts tax law, but it would be more appropriately located in a container holding Boston tax law, with only a pointer to that location residing in the Massachusetts tax law container. In this example, such an analysis could be accomplished by comparison of zip code information in the space registers, or logical rules-based analysis, with "state" being a larger set than "city". Or, i.e., operating under the registers of time relevance, the curve of time relevance for a concert might follow an ascending curve for the months prior, hit a brief plateau, and then reach a precipitous decline, at which time certain pertinent information only might be moved to an archival container of city events or rock concerts of that year. In this example, once the curve is mapped into a register, that map would cause an increasing frequency of pointers to that container in other containers or gateways, or inclusion of that container in other containers, as the analysis engine compares that curve with increasing user inquiry.

Referring now to Fig. **13 D**, a flowchart of the reconstruction process is shown.

Information containers **505** and **909**, residing within container **908**, operating under the rules governing container interaction within that container **908** downloaded **888103** to container **505** and **909** from gateway **9081** upon their entrance to container **908,** which rules had been downloaded **888102** from execution engine **500** acting under the direction **888101** of analysis engine **400**, and under the rules programmed into their own registers **404120, 909120**, compare **888104** the specified (by those rules) set of registers **404120, 909120**, i.e., time and space, and determine **888105** a container **404** encapsulated within **505** would be more appropriately encapsulated within container **909**.

Container **404** is moved **888106** to reside with container **909**. As the container **404** exits container **505**, the gateway of container **505**, being gateway **5051**, operating under the rules governing container interaction with a gateway **5051** upon egress or egress as programmed in the gateway editor **210** and modified **888108** by the execution engine **500** executing the instructions of the analysis engine **400**, or any greater logical analysis engine **408** providing execution instructions **888107** to an execution engine **508** operating in a larger encompassing container **108** entering through that container's gateway **208** or an independent gateway **707**, or sub-analysis engine operating at any gateway level, records **888109** the register information of container **404,** and alters the register information of container **404.** The gateway **5051** reports **888110** the transaction to the gateway **9081** of container **908**, being the next higher logical container. Gateway **9081** holds **888111** in gateway storage **205** the information until collected by one or more data collection processes **700**, or reported to one or more data reporting processes **600**, serving **888112** one or more analysis engines **400** residing independently on the system **10** or an analysis engine at higher logical container **303**. The analysis engine **400,** comparing **888114** reports of user hierarchically graded usage on information container **808** under the operations of the search engine **320** and the search interface **300**, after receiving **888113** reports from the data reporting means of container **404** being moved to container **909**, determines **888115**, i.e., that the number of time and space relevant containers residing within container **909** is sufficient to warrant an action, and directs **888115** the execution engine **500** to copy container **909**, nested within container **908**, to a third information container **808**. As the copy instruction from execution engine **500** transits the gateway of container **908**, the gateway **9081** records **888116** the instruction. The copy instruction interacts **888117** with the registers **909120** of container **909** regarding the rules governing its copying to another location. Once approved **888118** by the governing rules of registers **909120** appended to container **909**, container **909** is duplicated **888118**. As the duplicate container **909** exits the container **908**, the gateway records **888119** the register information **909120** of container **909**, and the registers **909120** of container **909** are altered **888120** by special instructions from gateway **9081** under the rules residing in gateway **9081** regarding ingress and egress and the rules residing in the registers **909120** of container **909** regarding alteration by gateways upon ingress and egress. Passing through independent gateway **707**, the register information **909120** is recorded **888121**, and awaits **888122** data collection or reporting **700, 600.** As container **909** enters container **808**,