

# Business Integration Using State Based Asynchronous Services

Alan McNamara, Badja Consulting, Australia  
Dr. M. Ali Chishti, Defence Housing Authority, Australia

## ABSTRACT

Architectural principles promote the use of interfaces—a facade behind which the implementation can be hidden. Current Service Oriented Service (SOA) analysis methods are based on providing synchronous services to expose transactional processes. However, this analysis method does not give a robust model for services implemented through workflow.

A business integration interface (façade) will need to:

- hide process implementation
- allow long running processes
- allow tracking of process status
- allow additional data to be provided when required
- handle events, regarding supplier or requester of a service
- manage exceptions

The service model presented in the OASIS ASAP [1] standard provides a good basis for the business integration interface, but the state based service model is not fully defined. This paper provides an analysis of the requirements for business integration services for long running processes, and proposes how the service model presented in the ASAP standard can be extended to satisfy these requirements. Practical examples from industry are used to illustrate these points.

## INTRODUCTION

The processes that provide business services necessarily look different from the outside than when viewed from the inside.

Consider the operations of a distributor. Their processes revolve around filling orders, maintaining stock levels, sending out bills (and ensuring they are paid) and paying their own bills. The processes to fulfill an order may involve checking the credit rating of the customer, checking stock levels, organizing transport, tracking delivery. In some cases, the goods can be supplied from current stock, in others they may need to be sourced from suppliers. A single order may have a combination of both. Some of the fulfillment processes may be internal, but some may be external—either outsourced or externally supplied.

But consider the same business function as viewed by the retailer who has ordered the goods. Many of the process steps do not concern them, only the outcomes. They are concerned about when the delivery will be made, being informed if delays occur, and being able to track their order—Is it still in the warehouse or in the hands of the carters? They are concerned about stock needing to be sourced from suppliers in terms of timeframes (especially if sourced from overseas or made to order), and ensuring the order has not been 'lost' or delayed somewhere in the process.

This paper takes the position that the customer's transaction and information requirements should be the basis of the interface between the parties. The complexity of the process which provides the asynchronous business service must be hidden behind a Façade [2]. Further, this paper describes how a state based model forms the best basis for that façade.

### BUSINESS INTEGRATION

The unit of business interaction is a business service—for example, a purchase. This may appear to be stating the obvious, but it is important to note that the business interaction is a *purchase*, and not *processing a purchase order*.

The business interaction has at least two parties—a service provider and a service consumer. These roles are asymmetrical for each interaction (e.g. the roles of buyer and seller in the purchase business service), but a party may have different roles in different interactions. For example, the distributor or wholesaler may be a seller in one interaction and a buyer in another.

A business service is asynchronous. It takes time to fulfill the business service, and the activities of the service consumer do not stop while awaiting the delivery of the service outcome. Note that saying the service is asynchronous is not tied into the form of the messaging between the parties—this could be synchronous or asynchronous as required,

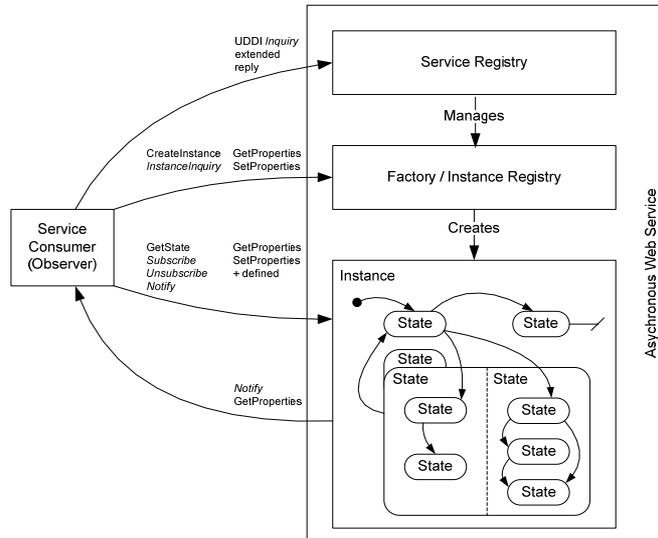
It is also important to consider the semantics of the service. In view of the customer, the 'Purchase' business service could be considered completed once the purchased goods or services have been delivered, while the supplier would only consider the interaction completed once both supply and payment have been completed. However, this may be better managed by the Purchase service spawning off the Payment process—as when payment may be through, for example, lease payments to a third party.

### STATE BASED ASYNCHRONOUS BUSINESS SERVICE

The Asynchronous Service 'Process Façade' can be represented the set of services shown in Figure 1.

This model is based on ASAP/WfXML [1,3], and has the same purpose—to allow business processes to work together. A number of significant changes have been made however, most based on the philosophical change from WfXML's desire to *expose* workflow for integration, to this paper's proposal to *hide* the workflow behind a Façade.

Most notably, unlike the Wf-XML model, there are no activities accessible by the service consumer. These activities can exist in the implementation, but are hidden from the Service Consumer.



**Figure 1. Service Model Overview**  
 [Statechart shown is for illustrative purposes only.]

**Service Registry**

The service Registry for an asynchronous web services must describe the interfaces for both the Factory and Instances, and the state structure.

The model for the Instance is a UML Statechart [4]. The instanceDetails definition will therefore need to capable of describing the Statechart state-transition model. Further, each state has its own interface model:

- Events that the State recognizes;
- Properties (including type definitions), that can be set and un-set;
- Synchronous services which are valid if the Instance is in that state.

Utilising the registry enquiry function, the Service Consumer has access to the URI location of the Factory, and the description of the state model.

**Factory**

Instances are created by the Factory, and the URI of the Instance returned to the Service Consumer. The factory also has properties which can be viewed and set.

The Service Factory also acts as an Instance Repository. As a factory could have significant numbers of instances, an inquiry function needs to be available, at a minimum being able to filter on the instance status.

**Instance**

The Instance may act as either or both a NotificationConsumer and NotificationProducer [5].

Note that the service consumer cannot change state directly—only by changing properties or notifying the instance of an event. These may trigger a status change, depending on the transition rules.

**Service Consumer (Observer)**

The Service Consumer may act as either or both a NotificationConsumer and NotificationProducer.

SUPPLY CHAIN

**Description**

Consider the main actors in a simplified supply chain:

- **Stores**
- **Transport Services**
- **Warehouses**
- **Suppliers**

In our example situation, the Stores and Warehouses, and some (but not all) of the transport is managed by the same organization. Sometimes Suppliers deliver directly to the stores, however it is preferred to delivery to the warehouse to aggregate many smaller loads and so reduce congestion of the Store loading docks.

A Store stocks a number of **Items**. They maintain stock levels appropriate to the sale levels and expected delivery times, as part of the **Stock Management** process. To maintain this level, they need to order more stock items—but Stores do not care if this needs to be purchased and supplied by a Supplier, or supplied from the warehouse.

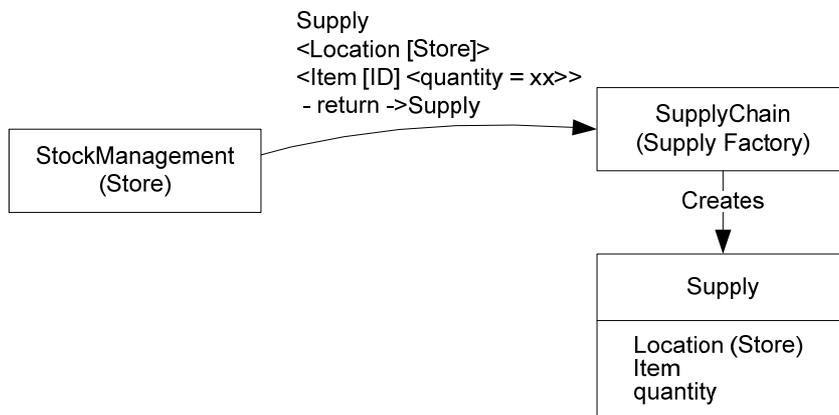
Supply Chain

One of the best principles for any architecture is separation of concerns— isolation between systems so that changes to one system have the minimum possible effect on any others. We therefore introduce an intermediate entity to hide the manner of how supply is achieved:

- **Supply Chain**

The Supply Chain accepts a store order from the Store, and then decides if this will result in a supply from a Supplier or from a Warehouse, or potentially, from another Store.

To correctly model this interaction as a process, rather than a document, we will use the verb **Supply**.



**Figure 2. Creation of the Supply Process**

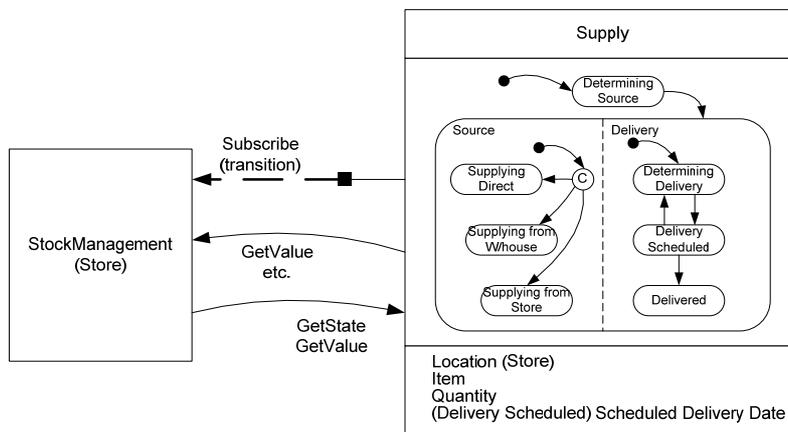
The Stock Management process asks the Supply Chain to Supply a certain quantity of a Item. This interaction creates the Supply process, and the Stock Management requester is given a reference to this process.

### Supply States

The supply process now needs to be monitored and supported. The Supply process is seen by the Stock Management process as a series of states, initially:

### DeterminingSource

Implied in creation of the Supply process is the subscription of the Stock Management process to the state changes in the Supply. The Supply service notifies the Stock Management when state changes occur.



**Figure 3. Monitoring and Servicing the Supply Process**

The Supply process may need further information from the Stock Management process. This may be information or other functions.

Some Supply parameters are valid only for certain contexts—defined by the hierarchical states. For example, the Expected Delivery Date is only valid for Delivery Scheduled state.

### Determining Method

The first state for the Supply process is determining the source of supply.

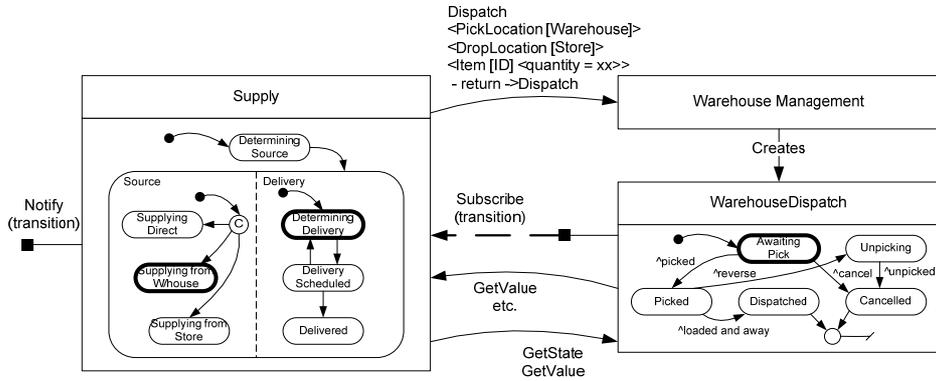
Looking now at a more specific example, the business rules in the Supply process may say that, for the selected product, the supply should first attempt to source from the local Warehouse.

The Supply process will attempt to start the dispatch process in the local warehouse. The Warehouse Management service will first check that either sufficient stock is on hand, or that stock can be sourced.

If the return to Supply service is OK, then the method of ‘Supply from Warehouse’ is selected, and the transition is made to:

SourceSupplyingFromWarehouse || Delivery.DeterminingDelivery.

Separate states for the supply source is used (rather than using a parameter) so that further sub-state modeling can be used.

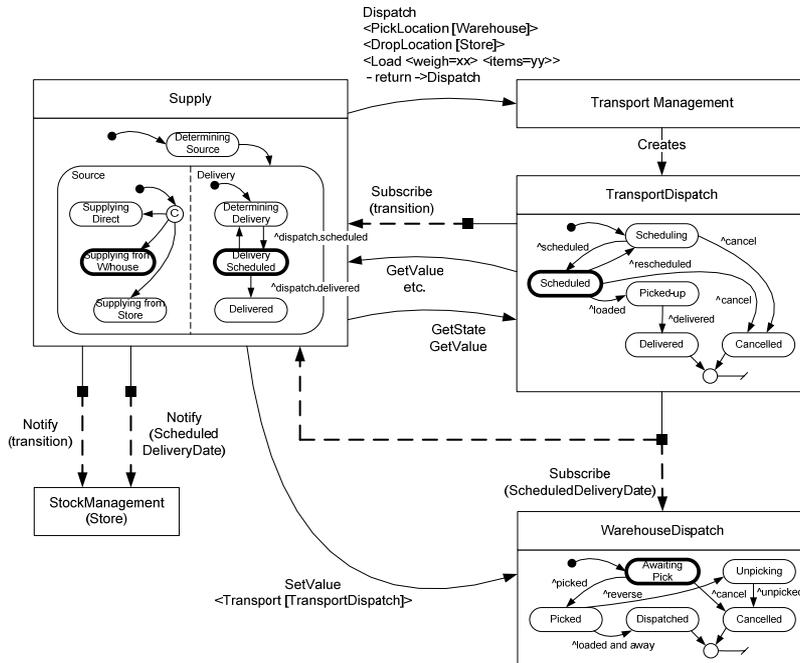


**Figure 4. Obtaining Warehouse Dispatch**

The Stock Management process is notified of this transition. Further, at any time, the Stock Management process can get the state of the Supply, and check on values of parameters that are valid for that state.

**Determining Delivery**

Once the Warehouse process is accepted as initiated, the transport can be arranged. A request for a Transport Dispatch is initiated, and scheduled.



**Figure 5. Obtaining Transport Dispatch**

The Supply process receives the notification of the scheduling, which results in the transition of Delivery to Delivery Scheduled. The Warehouse Dispatch process is also notified of the Expected Delivery Date, so it can plan its picking process.

### Re-scheduled Delivery

The discussion so far demonstrates how the various processes can be linked, but does not show how the state based models are superior to process step models.

If we now take the example where the Transport Dispatch fails—the scheduled pickup did not occur.

The Transport Dispatch process will be monitoring the dispatch against the schedule and notes the pickup did not occur. There is a choice between cancellation and rescheduling the Transport Dispatch—the operator decides to reschedule.

The state transition on Transport Dispatch triggers a notification to the Supply process. This triggers the Delivery state change from Delivery Scheduled back to Determining Delivery. In this case, the Supply operator decides not to override the reschedule with a cancel. The Stock Management is of course also notified that the state has dropped back to Determining Delivery.

In a slightly different case, consider if the Transport Dispatch Process has a policy that says it will only dispatch a truck for delivery if the items are already picked and waiting. In this case, Transport Dispatch Process may subscribe to the Warehouse Dispatch process, and reschedule if not Picked in time.

The Transport Dispatch process does its reschedule, and transitions to Scheduled. The both the Warehouse Dispatch and Supply processes will be notified of the new Scheduled Delivery Date, which is passed on by the latter to the Stock Management process.

Note how the Supply service acts as a façade for the Stock Management process—it only sees the Scheduled Delivery Date from the Supply service, and doesn't need to track or care about the transport or warehouses, only on the information it deems necessary to track.

### HOUSING SERVICES

#### **Description**

Defence Housing Authority Australia [DHA] is a multi-billion dollar government business enterprise. It provides housing and relocation services to the members of other Australian public sector organizations. Since these members are the end-users of the services offered by DHA they are considered as **Customers** while their organizations are considered as **Clients**.

For this discussion DHA's Customer Service model is simplified into following business processes:

- Relocations Management—a process that involves point to point relocation of Customers as well as their family and dependents.
- Accommodation Management—a process that is mainly focused on provision of housing in the new location [**gaining location**].
- Property Management—Management of property stock that is controlled and managed by DHA.

In past three years DHA has reengineered its business processes to radically improve in the key process-performance measures such as cycle-time, cost, quality and customer-responsiveness. Reengineering effort was also aimed at overhauling the service delivery model and the information technology based

business solutions. Before the overhaul DHA's business solutions were composed of numerous islands of automation that supported fragments of organizational functions. Integration between the solutions was limited and was devoid of important systemic qualities such as flexibility and extensibility. As a result staff had to rely on number of manual procedures and work-arounds to provide services to Customers.

While the entire application architecture was revisited during the reengineering exercise, this discussion is primarily focused on the redesign of the Accommodation (Management) Process.

ASAP model was used to design the underlying application services due to the fact that DHA's business processes often: have a longer life span; use complex rule-base and need to be flexible and responsive to the frequent changes in Customers' personal circumstances.

Some adjustments are made in the solution presented here for the purpose of theoretical generalization in-line with the model proposed in the earlier section however, the discussion is well grounded in the actual implementation. Also some elements of the workflow (including exception- and alternate-flows) have been either simplified or excluded in order to manage the scope of discussion.

### **Functional Overview**

Following services were designed to perform specific functions within the Accommodation workflow:

1. **RelocationService**—to manage the lodgment-of, and processing of Application for Relocation [AFR]. This service primarily supports Relocation Management and discussed here as a starting point of the occupancy and vacancy cycle of DHA's properties.
2. **OccupancyService**—to be used by RelocationService in determining a housing solution for a relocation.
3. **AllocationService**—to be used by OccupancyService for searching and selecting a house from DHA's property stock.
4. **VacancyService**—to be used by RelocationService to logically release a previously occupied property back into *available to occupy* housing stock.
5. **PropertyService**—to keep a record of current state of all properties managed by DHA in order to support Accommodation workflow as well as property life-cycle processes such as property maintenance and property portfolio management.

RelocationService is often initiated upon receiving a request for relocation from the Customer or Client. Once RelocationService establishes a valid case for relocation and determines the relocation and housing entitlements it initiates the OccupancyService and VacancyService in order to identify and hold an accommodation in the gaining location and to vacate a property in the current location [**losing location**] respectively.

Business rules built into OccupancyService determine the source of new accommodation e.g. property from DHA's housing stock [Service Residence or SR] or otherwise. If a Customer is entitled for a SR, OccupancyService creates an instance of the AllocationService to search for a SR (that is currently

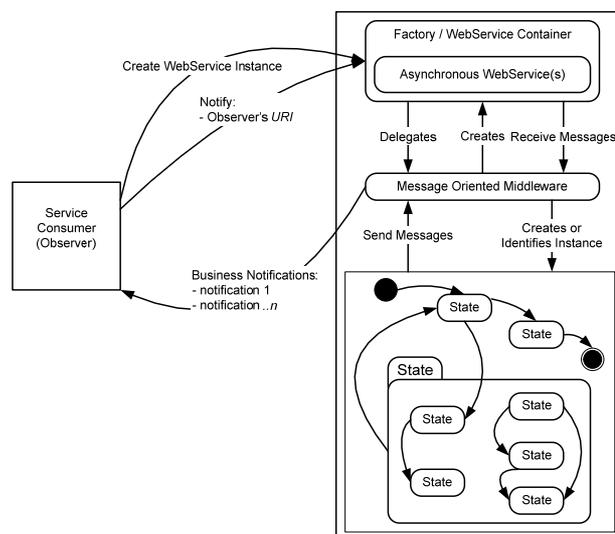
available or will be available in the right timeframe) in the gaining location according to Customer's entitlement.

If one or more properties are found, AllocationService allows Customer to choose a property of his / her liking from the list. This selection is termed as lockdown. Once a lockdown takes place all services directly involved in Accommodation workflow are directly or indirectly notified using a publish-subscribe arrangement. Upcoming and actual vacancies of properties are also managed and published using the similar arrangements. This design ensures that Customers' accommodation arrangements are made according to their entitlements and DHA's property stock is utilized to its full potential.

### Logical Implementation

Webservices based notification [WS-Base Notification][5] and ASAP [1] concepts are used in this implementation. Change of internal state [state-transition] of NotificationProducer is the primary trigger (or situation) for a majority of notifications however, not all updates between the Notification-Producer [Instance] and NotificationConsumer [Observer] are attached to state-transition(s).

The model proposed in this document is largely followed although as stated earlier this discussion has made some adjustments in the hindsight.



**Figure 6: Logical implementation of Accommodation Workflow services**

Right level of functional granularity is used while designing the relevant services this allowed separation of areas of concerns for each service and assisted in hiding the activities within a service behind a state-base façade.

A common interaction between observer and (service) instance is based on a `HttpRequest` for an asynchronous webservice that works as an instance factory and provides a reference of that instance to the subscriber.

A message oriented middleware [Message Queue or MQ] is used as a delegate for all incoming and outgoing notifications. As a result when a request for a service instance is received by the asynchronous webservice, Message Queue creates the instance with the parameters provided by the observer and sends a reference to the instance back to the observer via asynchronous webservice. When observer needs to update the instance a similar channel is followed. However when instance need to notify the observer, Message Queue directly invokes the observer's webservice on the behalf of instance. This implementation model did not require a UDDI since URIs of all webservices in the solution were already known. Figure 6 delineates the implementation model.

### **RelocationService**

`RelocationService` supports the process of managing point to point relocation of DHA's Customers. An instance of `RelocationService` can have a life-span of a fortnight to six months. During that life-span it goes through number of transitions and states. There follows a simplified version of `RelocationService`' key states (that excludes the states in alternate and exception flows):

1. *Relocation Initiated*—A valid case for relocation has been established.
2. *Application Data Entered*—Information supplied by the Customer in the application for relocation has been captured including key dates such as: Expected Vacancy Date of the current residence and Expected Occupancy Date of the Future Residence.
3. *Housing Solution Confirmed*—Future housing has been arranged either from DHA's housing stock or otherwise.
4. *Movement Plan Confirmed*—Arrangements for pickup and delivery of household inventory and temporary accommodation has been finalized.
5. *Financial Allowances Payment*—Allowances relevant to the relocation are paid to the Customer.
6. *F&E Uplift*—Freight and inventory is picked up from the current address.
7. *Current Accommodation Vacated*—Current accommodation is either vacated or its vacancy date is confirmed.
8. *New Accommodation Occupied*—Occupancy of the future accommodation is confirmed i.e. future address becomes the current address.
9. *F&E Delivered*—Freight and Inventory is delivered to the new address.
10. *Relocation Confirmed & Closed*—Customer has been successfully relocated into the new location.

### **Occupancy Cycle**

Since a housing is needed in the gaining location [step 3] with almost every relocation request `RelocationService` asks the `OccupancyService` to arrange for such housing regardless of its source i.e., either from—or outside of—DHA's housing stock. With this request `RelocationService` passes the necessary details about the Customer, gaining location, housing entitlements of the Customer and the Expected Future Occupancy Date. `OccupancyService`

then creates a reference for this request and keep RelocationService upto-date about the progress made in arranging the housing for an AFR. Housing arrangement is referred to as '**housing solution**' in the Accommodation workflow.

### **OccupancyService**

OccupancyService contains two subsets of states i.e., HousingSolutionSource and LockdownStatus. HousingSolutionSource represents the type of the housing solution (that is determined by the business rules) while LockdownStatus represents the key states of progression in finding the housing solution.

HousingSolutionSource could have one of following states:

- *Own Means [OM]*—Customer will arrange his / her own accommodation and no assistance is required from DHA in either finding or paying for that accommodation.
- *Own Home [OH]*—Customer has a property in the gaining location that will be used as a housing solution.
- *Living In Accommodation [LIA]*—Customer will be accommodated on his / her actual posting site [e.g. Defence Base].
- *Service Residence [SR]*—Customer will be accommodated in one of DHA's property in the gaining location.
- *Rental Assistance [RA]*—Customer will find a rental property and DHA will contribute in the rent payments.

LockdownStatus states, on the other hand, translates into the following:

- *Searching*—This state represents the fact that OccupancyService has requested the relevant '*property search service*' (e.g. AllocationService in case of SR) to find a housing solution according to Customer's entitlements. This mainly applies to search for SR using AllocationService at this stage however this design allows for additional functionality to search for other types of housing solution.
- *Unlocked*—This state indicates that a property search service (AllocationService) is awaiting for the Customer to select [lock] a property from a subset of available to occupy properties in the gaining location. This state is also more meaningful in SR context.
- *Locked*—It is a generic state that suggest that a property has been selected as a housing solution. A property address is needed (among other information) before a transition is made to Locked state.
- *Occupied*—Also a generic state that is reached once Customer actually occupy the housing solution and RelocationService notifies the OccupancyService with the Actual Occupancy Date.

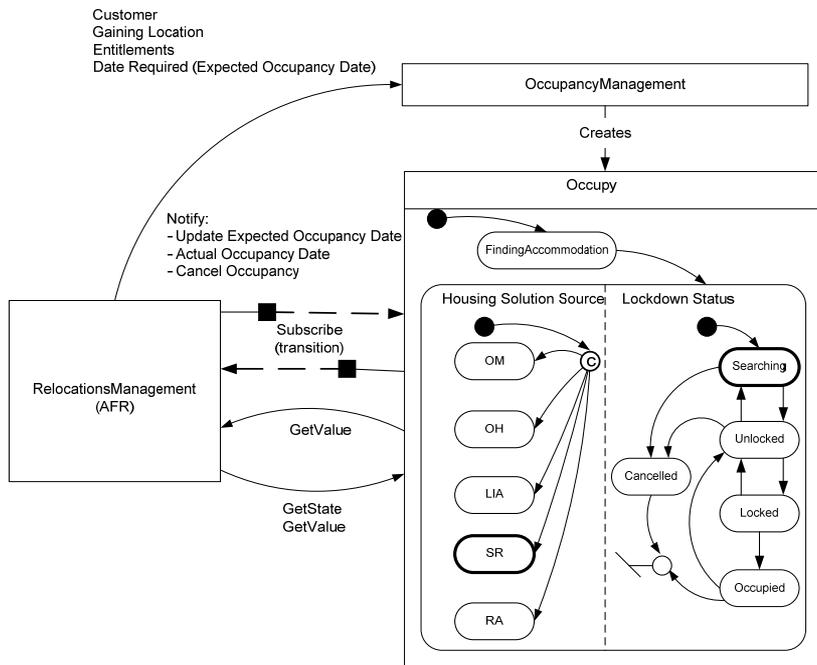
After initiation HousingSolutionSource and LockdownStatus together defines the current composite state of the OccupancyService. For example, in a prevalent business scenario, where RelocationService requests a housing solution for a Customer and business rules determines that the Customer is entitled for a SR, OccupancyService makes a transition from initial state of FindingAcommodation to the following state and notifies RelocationService:

HousingSolutionSource.SR || LockdownStatus.Searching

This state represents the fact that Customer's housing entitlement has been determined and OccupancyService has invoked the AllocationService to find

a SR in the gaining location according to Customer's entitlement. Once AllocationService indicates that one or more SR exists in the gaining location (and a Customer login has been created) OccupancyService makes a transition to following state and notifies the RelocationService:

HousingSolutionSource.SR || LockdownStatus.Unlocked



**Figure 7: Obtaining a HousingSolution**

In a situation where a property cannot be found according to entitlement DHA staff can intervene and allow OccupancyService to offer properties that are outside of Customer's entitlement by altering the search criteria. This also assist DHA staff to reserve / un-reserve a property as well as to offer a particular property to the Customer (these activities are still represented by Searching and Unlocked states and Customer still have to lock a reserved or specially offered property using the AllocationService).

Next, Customer selects the property using AllocationService which results in notification to OccupancyService with necessary details. This allows OccupancyService to make a transition to the following state:

HousingSolutionSource.SR || LockdownStatus.Locked

OccupancyService now notifies the new state to RelocationService, provide the details of HousingSolution and awaits for notification from RelocationService for the Actual Occupancy of the housing solution. Once actual occupancy date becomes known transition to following state is made and both AllocationService and RelocationService are notified with the new state:

HousingSolutionSource.SR || LockdownStatus.Occupied

OccupancyService is also notified by the RelocationService about updates to expected and actual occupancy dates and cancellation of a Relocation. While changes in expected occupancy dates have little impact on OccupancyService, cancellation results in moving the LockdownStatus back to Unlocked state.

## AllocationService

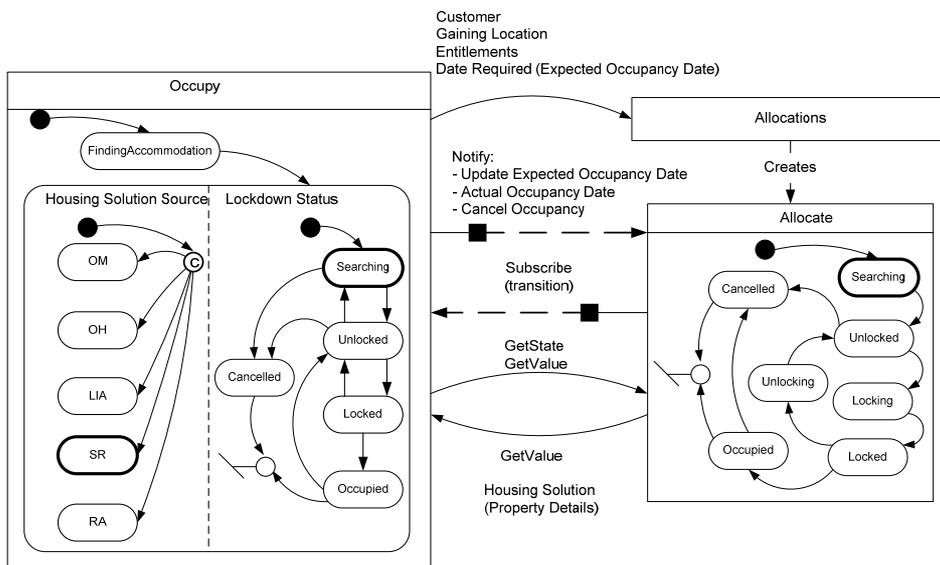
AllocationService is also a state based service that performs the following tasks:

- Search for SR in gaining location using the search criteria as notified by the OccupancyService.
- Create login credentials and notify the Customer if one or more properties are found.
- Let Customer login and lock a property.
- Unlock a property upon instructions from OccupancyService.
- Notify OccupancyService & PropertyService with key states and property information.

Following is the description of the key states in the AllocationService:

- Searching—DHA’s property stock is being searched according to search criteria provided by the OccupancyService.
- Unlocked—A property has not been selected yet or the selection has been reverted.
- Locked—Customer has selected a property.
- Occupied—Selected property has been occupied.

As stated earlier AllocationService is an implementation of a ‘*property search service*’ and it only assist in searching, locking and unlocking SR. When OccupancyService request a search for a SR in gaining location an Allocate object is created with the search parameters provided by the OccupancyService.



**Figure 8: Allocating a Property**

If one or more SR are found according to search criteria AllocationService makes a transition to the Unlocked state and notifies the OccupancyService. At this stage a Customer login is also created and he / she is notified.

Next Customer login into the AllocationService and selects a property. This results in transition to Locked state. At this point both PropertyService and OccupancyService are notified with the change in state and housing solution details.

Similarly when OccupancyService notifies about a cancellation, AllocationService makes a transition back to Unlocked state and subsequently to Canceled state.

Notification of Actual Occupancy Date results in transition to Occupied state which in turn triggers the similar notification to the PropertyService.

### ***PropertyService***

PropertyService performs following specific tasks:

- Storing and maintaining addresses of all properties that exists within Accommodation workflow regardless of their source i.e., owned by DHA or otherwise.
- Maintaining Occupancy / Vacancy Details of all properties that are managed by DHA including those that are on lease from external sources.
- Notifying other subscriber services whenever a property status is changed including those services that are relevant to other workflows such as maintenance, estate & portfolio management, and accounting & financial management.

From Accommodation workflow viewpoint property status is represented by two logical states—available to occupy [VAV i.e. VOID Available] or unavailable to occupy [VUN i.e. VOID Unavailable]. Sub-states are used to further explain the property status. For example a property which is Vacant but unavailable could be in such state because it has been allocated to a Customer who is expected to occupy the property on a particular date. This information could easily allow the time window for periodic maintenance to be completed. There follows a brief description of the composite states in PropertyService:

- LET || OCC (Let / Occupied)—Property is allocated and is occupied by the Customer.
- VAV || FUTV (Vacant Available / Future Vacant)—Property is either available now or its availability in future is known.
- VUN || FVA (Vacant Unavailable / Future Vacant Allocated)—Property is vacant and has been allocated and expected occupancy date is known.
- VAV || VCAP (Vacant Available / Vacant Property)—Property is available for allocation and is currently vacant.
- VUN || VA (Vacant Unavailable / Vacant Allocated) Property is not available because it has been allocated.

AllocationService notifies PropertyService with all transitions.

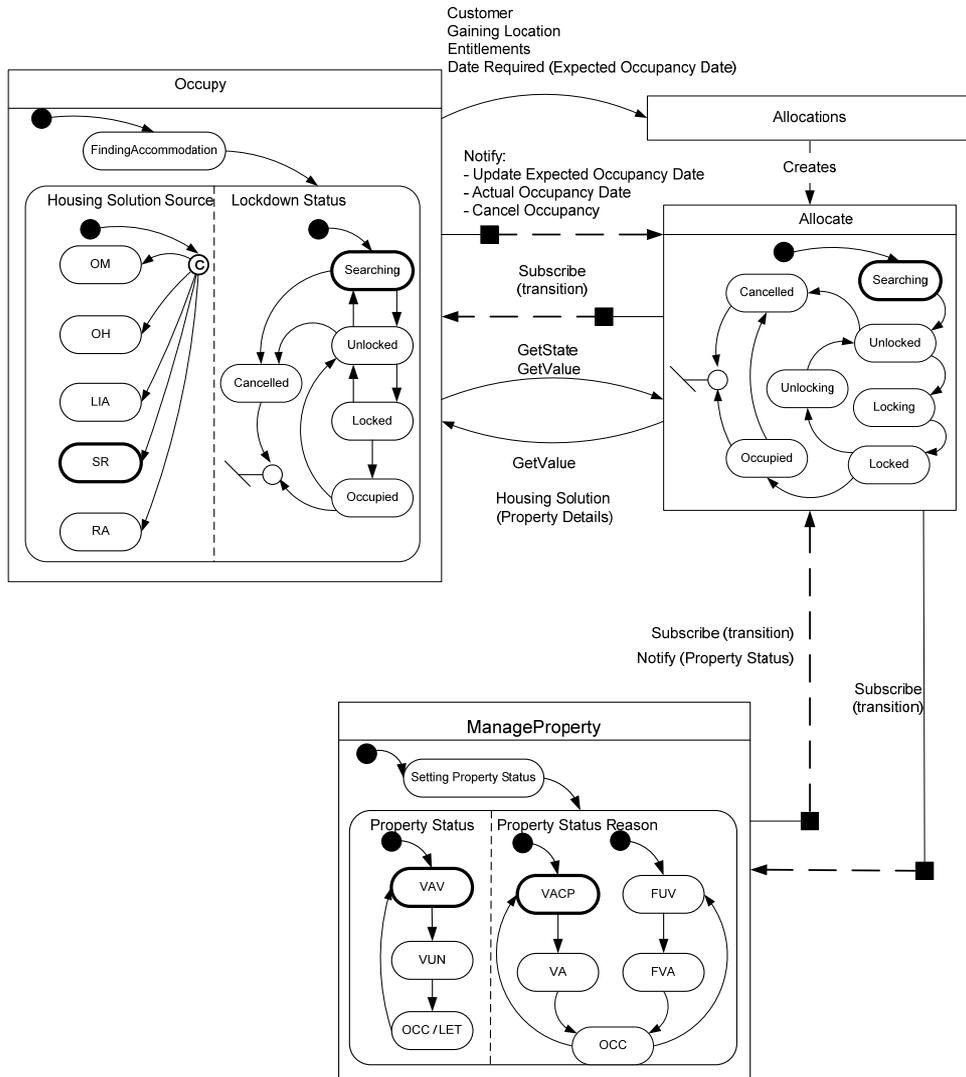
Within accommodation workflow, PropertyService acts in both roles as an observer and as an instance. As an observer to AllocationService and VacancyService it receives notifications for all status changes as well as information about housing solution including dates for actual and expected future occupancy and vacancy.

Following table provides the mapping between the key states of OccupancyService, AllocationService and PropertyService.

<b>OccupancyService</b>	<b>AllocationService</b>	<b>PropertyStatus    StatusReason</b>
HousingSolutionSource.SR    LockdownStatus.Locked	Locked  (Create Future Occupancy of a particular SR)	<ul style="list-style-type: none"> <li>• If property status is VAV with property reason of FUTV then set property reason to FVA.</li> <li>• If property status is VUN then set property reason to FVA.</li> <li>• If property status is VAV with property reason of VACP then set reason to VA.</li> <li>• If property status is VUN then set property reason to VA.</li> </ul>
HousingSolutionSource.SR    LockdownStatus.Occupied	Create Actual Occupancy	<ul style="list-style-type: none"> <li>• Let / OCC</li> </ul>
HousingSolutionSource.SR    LockdownStatus.Canceled	Canceled  (if there was no actual occupancy date supplied by AllocationService it will be considered by PropertyService as a request for canceling a future occupancy)	<ul style="list-style-type: none"> <li>• If property status was VUN and property reason is FA then set property reason to FUTV.</li> <li>• If property status was VUN and property reason is VA then set property reason to VACP.</li> </ul>
HousingSolutionSource.SR    LockdownStatus.Canceled	Canceled  (Cancel Actual Occupancy)	<ul style="list-style-type: none"> <li>• VA</li> <li>• VACP</li> <li>• TUAV (Temporary Unavailable)</li> <li>• PUAV (Permanently Unavailable)</li> </ul>

**Table 1: Mapping between the key states of RelocationService, OccupancyService, AllocationService and PropertyService**

Figure 9 shows the information flow for the occupancy cycle:



**Figure 9: The Occupancy Cycle**

**VacancyService**

VacancyService, as the name suggest is used by RelocationService to (logically) vacate a housing solution. When a Customer is expected to move out of a current residence a Future Vacancy needs to be created. RelocationService requests the VacancyService to vacate the housing solution. VacancyService creates a Vacate instance and provide a reference of that instance to RelocationService.

Again sub-states are used to fully represent the current state of VacancyService.

HousingSolutionSource sub-states are used to identify which type of housing solution is being vacated while VacancyStatus sub-states are used to represent the progress made.

For example when business rules within VacancyService identifies that RelocationService has requested a vacancy of SR and has provided the Expected (Future) Vacancy Date it makes a transition from its initial state to the following and notifies the RelocationService with the new state:

HousingSolutionSource.SR || VacancyStatus.Vacating

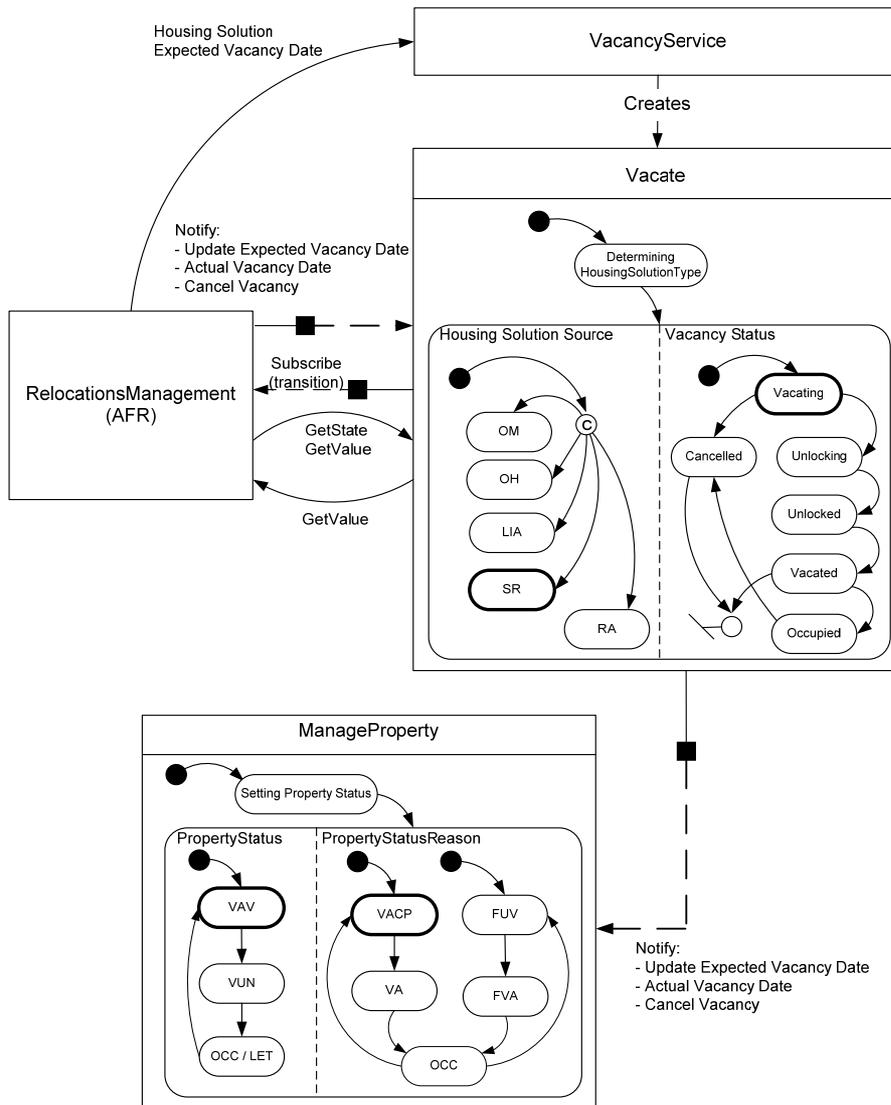
Since PropertyService has an interest in future vacancies and has a subscription to VacancyService it is also notified with the new state and the Expected Vacancy Date.

Similarly when RelocationService notifies the Actual Vacancy Date, VacancyService makes a transition to HousingSolutionSource.SR || VacancyStatus.Vacated and notifies both the RelocationService and the PropertyService.

Table 2 provides the mapping between the key states of VacancyService and PropertyService while Figure 10 provides the details of interactions between RelocationService, VacancyService and PropertyService.

VacancyService	Implications for PropertyService States	
	Property Status	Property Reason
HousingSolutionSource.SR    VacancyStatus.Vacating <i>(Create Future Vacancy)</i>	VAV	<ul style="list-style-type: none"> <li>FUTV—Future vacant</li> </ul>
HousingSolutionSource.SR    VacancyStatus.Vacated <i>(Create Actual Vacancy)</i>	If a property was not locked down then set the status to VAV else set it to VUN.	<ul style="list-style-type: none"> <li>If property was not locked down set reason to VACP.</li> <li>If property was locked down and future occupancy date does exist then set reason to VA.</li> </ul>
HousingSolutionSource.SR    VacancyStatus.Canceled <i>(Cancel Future Vacancy)</i>	LET is set to previous status.	<ul style="list-style-type: none"> <li>OCC is set to previous reason.</li> </ul>
Notification of new Future Vacancy Date <i>(Update Future Vacancy)</i>	No change to status.	No Change
Notification of change in Actual Vacancy Date <i>(Update Actual Vacancy)</i>	No change to status.	No Change

**Table 2: Mapping between the key states of VacancyService and PropertyService**



**Figure 10: The Vacancy Cycle**

RELATIONSHIPS TO EXISTING STANDARDS

**ASAP / WfXML**

The concept of the state-based, asynchronous model is primarily based on the OASIS ASAP / Workflow Coalition Wf-XML models [1,3]. The primary difference is that the interface is designed specifically as a façade to be presented to external parties, rather than managing the process itself.

In this role, the expressiveness of the state model becomes very important.

ASAP modeling [1] of the of the status focuses on the status of the work engine (state types):

**open.notrunning.suspended:** A resource is in this state when it has initiated its participation in the enactment of a work process, but has been suspended. At this point, no resources contained within it may be started.

**open.running:** A resource is in this state when it is performing its part in the normal execution of a work process.

**closed.completed:** A resource is in this state when it has finished its task in the overall work process. All resources contained within it are assumed complete at this point.

**closed.abnormalCompleted:** A resource is in this state when it has completed abnormally. At this point, the results for the completed tasks are returned.

**closed.abnormalCompleted.terminated:** A resource is in this state when it has been terminated by the requesting resource before it completed its work process. At this point, all resources contained within it are assumed to be completed or terminated.

**closed.abnormalCompleted.aborted:** A resource is in this state when the execution of its process has been abnormally ended before it completed its work process. At this point, no assumptions are made about the state of the resources contained within it.

The examples shown illustrate that these statuses are unimportant to the interface, and that the interface designer must be able to specify their own status models.

Further, these example shows that the interface status model must be as expressive as UML Statecharts [4]. In particular, the ability to express hierarchical and parallel states is very important

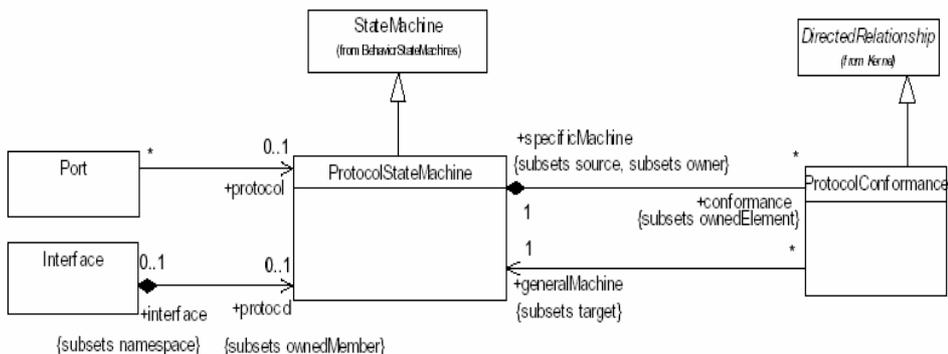
**UDDI and UML**

It is proposed that the service Registry for Asynchronous Web Services be an extended version of UDDI [6]. The UDDI entry should describe both the Factory and Instance interfaces.

The factory interface would be described in standard UDDI fashion—as these services are traditional synchronous web services. The instance interface, however, requires extension to the UDDI instanceDetails.

The UML 2.0 specification [7] supports the modeling of an Interface as a ProtocolStateMachine.

*Package ProtocolStateMachines*



**Figure 11: Protocol State Machines**

Serialisation of this model using XMI [8] may be a suitable base for defining the StateChart and associated Interface as a ProtocolStateMachine.

### CONCLUSION

This paper proposes a new approach in considering asynchronous web services. In particular, it is proposed that the model for asynchronous services should stress the state nature of the façade presented to consumers, and to hide the process supporting the service.

Further work is required to ensure the specification of ASAP and UDDI standards can be modified to properly support this new approach. Standards based on UML are preferred.

### REFERENCES

- [1] OASIS (2005) Asynchronous Service Access Protocol (ASAP) Version 1.0 Proposed Committee Draft, May 18, 2005
- [2] Gamma, E et al (1994) "Design Patterns—Elements of Reusable Object-Oriented Software." Addison Wesley Longman: Reading Massachusetts).
- [3] Workflow Management Coalition (2004) Wf-XML 2.0—XML Based Protocol for Run-Time Integration of Process Engines, Draft October 8, 2004.
- [4] OMG (2003) UML 2.0 Infrastructure Specification OMG Adopted Specification ptc/03-09-15.
- [5] OASIS (2005) Web Services Base Notification 1.3—(WS-BaseNotification) Public Review Draft 01, 07 July 2005
- [6] OASIS (2004) UDDI Version 3.0.2 Spec Technical Committee Draft, Dated 20041019.
- [7] OMG (2005) Unified Modeling Language: Superstructure version 2.0 formal/05-07-04
- [8] OMG (2005) XML Metadata Interchange (XMI) Specification May 2005 Version 2.0 formal/05-05-01