HL7 Service Oriented Architecture

Special Interest Group (SOA SIG)

SERVICE ORIENTED ARCHITECTURE AND HL7 V3

MESSAGING TO SERVICES MIGRATION METHODOLOGY

VERSION NO. 1.0

DATE: January 17, 2007

Editors	Alan Honey: alan.p.honey@kp.org (Kaiser Permanente) Brad Lund : brad.lund@intel.com (Intel Corporation)
Authors	Alan Honey: alan.p.honey@kp.org (Kaiser Permanente) Ani Dutta : ani.dutta@va.gov, ani.dutta@siliconspirit.com (Veterans Health Administration / Silicon Spirit) Jari Porrasmaa : Jari Porrasmaa <jari.porrasmaa@uku.fi> (University of Kuopio / HL7 Finland) Juha Mykkanen : Juha Mykkanen <jumykkan@messi.uku.fi> (University of Kuopio / HL7 Finland) Kathleen Connor: kathleen.connor@foxsys.com (Fox Systems) Manoj Kumar: Manoj.Kumar@bcbsfl.com (Blue Cross Blue Shield Florida) Rick Stevens: Rick J Stevens rstevens@us.ibm.com (IBM)

Preface

Notes to Readers

This document is a detailed but informal document, aimed at defining an approach for implementing healthcare services within a Service Oriented Architecture. This is intended to complement the Service Specification Framework (SSF) defined within the Healthcare Services Specification Project (HSSP), but provide an additional interim method of defining and implementing web services based on HL7 V3 artifacts.

Work is also ongoing to marry this work, and other related HSSP work with the overall HL7 processes (HDF etc.). As and when that work matures, this document will be updated and/or replaced.

Changes from Previous Release

This is the first public release of this document.

Acknowledgements

In addition to the listed authors, the following individuals are acknowledged for their contributions during the development and review of this document.

Ann Wrightson (CSW UK)

Ioana Singureanu (VHA)

John Koisch (OCTL Consulting)

Virinder Batra (IBM)

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

The purpose of this document is to describe a methodology for defining services within the healthcare domain, in particular, for areas covered by Health Level 7 (HL7) domain content; an effort known as Service Oriented Architecture for Health Level 7 (SOA4HL7)[1]. The methodology described herein is accompanied by a set of deliverables relating to infrastructure and architecture that collectively form the overall approach.

The document is particularly aimed at providing guidance that will help in the identification and enumeration of services based on existing HL7 messaging artifacts. The document aims to provide a practical approach, especially to existing HL7 committees interested in SOA. Following this methodology should lead to the production of appropriate (from a SOA perspective) service definitions, which may then be proposed as standards through the main HSSP process if this is appropriate.

The Healthcare Services Specification Project (HSSP) hosted jointly by the Object Management Group (OMG) and HL7 has created a methodology for defining industry standard services, known as the Services Specification Framework (SSF). The SOA4HL7 effort was chartered to complement their work by providing a method of defining interim services in a consistent fashion where formal standards have not yet been developed; recognizing the reality that software vendors and individual healthcare organizations are developing and implementing services today, without consistency or agreed methods.

Typically, methodologies consist of process, techniques, roles and artifacts (deliverables). Therefore, understanding that different types of organizations will define services for different purposes under various circumstances, rather than define a single, specific directive process, this document will concentrate on describing the artifacts and techniques. Specifically, it will focus on the artifacts that should be produced when defining services, guidelines for the services specification and techniques for design as well as a set of implementation considerations. Some process alternatives will be included for illustration purposes only. Where practical, the guidelines will reference existing HL7 artifacts which can be used as a basis for service definitions. Ideally, a simple deterministic mapping between existing HL7 artifacts and Service artifacts would be defined, but this will not provide appropriate SOA-friendly solutions in many cases. It is believed that the approach defined in this document will be the most valuable in terms of achieving the desired consistency. Further investigations will continue over time within HSSP based on experience.

[1] The SOA4HL7 project is being run under the auspices of the HL7 Services Oriented Architecture SIG as part of the larger Healthcare Services Specification Project (HSSP). http://hssp.wikispaces.com/

1.2 Background

The methodology described in this document is one of four main deliverables identified in the approved SOA4HL7 charter. An excerpt of the charter is shown below:

- **Architecture Requirements** - Define and agree on a set of architectural requirements that ensure Service Oriented Architecture (SOA) benefits can be realized and interoperability maximized (at least meeting minimum interoperability levels defined by HL7).
- **SOA Framework** - Based on the architectural requirements, define an SOA Framework/Approach that leverages existing and emerging IT standards to enable services to be consistently identified, described and used in healthcare environments.
  - This should provide a consistent technical context for HSSP OMG RFP submissions.
  - Include both a "generic" SOA approach that is not tied to specific technology and a specific technology implementation for web services[2].
- **Methodology** - Define extensions to the HSSP SSF methodology for creating service definitions and implementations, including approaches to conformance and profiling where appropriate. This should offer a consistent way to define and implement Services for HL7 (and other healthcare as appropriate) content. This is available at: http://hssp-infrastructure.wikispaces.com/
- **HL7 V3 Infrastructure Mapping** - Define a mapping of current V3 artifacts to the SOA framework. This should provide at least rules for deriving or transforming from SOA elements (contract or headers) to (at least) mandatory HL7 Wrapper items. This will include identification of those elements that should be left to other protocol and technology standards and any constraints that should be imposed on those elements.

In addition, the charter included the following clarification, which is relevant to this body of work:

> *"The main HSSP has defined a full methodology for specifying Services, starting with HL7 Service Functional Models and then OMG RFPs to provide fully specified interoperable service standards. This project will propose extensions to the SSDF (and other HL7 deliverables as appropriate) to enable interim service definitions to be defined in absence of fully specified HSSP services. (This can be seen as a "lite" process, which will complement and not replace the full HSSP process defined in the SSF). The purpose is to allow software vendors and healthcare organizations to define and produce services in a consistent fashion, in line with overall IT industry standards. In the absence of this, vendors are likely*

[2] The conceptual solution will leverage other work where possible to provide an abstract framework. Working within that model, or in parallel with it, we would define the solution based on the Web Services stack (i.e. XML, SOAP, WSDL, WS-* etc). It should be possible to subsequently define an implementation of other SOA technology stacks as and when required, but would not be included in the initial effort. The initial end game has to be an "implementable" solution, but also needs to provide a good conceptual foundation.

to continue to develop their own approaches to exposing service functionality where neither fully specified HSSP services nor a standard SOA framework for healthcare is available."

Appendix A contains a set of diagrams that depict the relationship between SOA4HL7 and the main HSSP SSF.

In the course of this work, the following questions need to be answered within the context of SOA:

- How should we define services within HL7 domain areas?
- SOA aims to enable dynamic business change and cost-efficient management of assets. How do we achieve the above and still maximize value from existing HL7 work?
- How do we define services that are appropriate and consistent?
- In what circumstances does it make sense to use services?
- How much should HL7 specify and what should be covered by "general" industry standards?
- How much should be left to individual organizations with respect to infrastructure?
- How should the healthcare specific and general IT standards work together, without creating too much dependency or coupling between them?
- From an architecture and infrastructure perspective, how do we specify enough to ensure interoperability and no more?

1.3 Scope

This document will ONLY cover the methodology and syntax (e.g. WSDL) to identify and define HSSP services and is specific to the health services domain; although it attempts to maximize the use of general IT industry approaches. Infrastructure related items i.e. technical architecture layers or HL7 transmission wrappers are beyond the scope of this document and will be covered by other HSSP chartered efforts.

It should also be noted that there are service specifications in progress within HSSP for four services that provide a range of general-purpose capabilities. One in particular, the Retrieve, Locate Update Service (RLUS), provides generic interfaces for locating, retrieving and updating medical records. Through the use of semantic profile mechanism (see SSF), this service may produce and consume many different forms of content, including HL7 V3 or V2 messages. Using this approach verses defining more specific, explicit services via this methodology, is an architectural and implementation choice that can be made in some circumstances (particularly for data oriented services). However, the architecture portion of the HSSP effort would still be relevant even in that case.

1.4 When to Use Services

From a standards perspective the answer is a great deal easier than from an individual project perspective. When should a standard service be defined? The simple answer is when sufficient organizations taking an SOA approach identify the requirement for the service. Certainly, interactive request reply scenarios make very good candidates, e.g. scheduling, eligibility checking, entity identification, demographics or other data look up and updates etc. It is however, worth considering the latter question, i.e. in what project situations should services be used rather than messaging, or vice versa.

One of the main points of SOA is to enable loose coupling which frees the Service Consumer from the details of the implementation of the Service itself. From a standards and interoperability perspective, this is an important aspect.

It must be stated that there is no simple answer or industry consensus to this question, other than “it depends”. Among others, the following factors should be considered:

- Is this a new development or legacy enhancement or integration effort? Are there already messaging solutions in place that work or existing services that can be reused?
- Is the organization committed to a standard architecture, e.g. there may be benefits from common security, monitoring, management perspectives?
- Is the business process/function volatile or stable? What levels of change are expected? How important is alignment between system functionality and business processes. Services that align with the business are claimed to provide better business agility.
- What technologies do the organization use and what is the direction?
- Is reuse important, and is the capability likely to offer great reuse potential? Services can provide high levels of reuse particularly where the clients are likely to be heterogeneous.
- Is there a need for interoperability and information crossing system and/or organizational boundaries?
- Are there likely to be very large volumes of data transferred between two systems on a frequent basis. This leads to a messaging or possibly batch/ETL solution.

Messaging approaches have been successfully implemented and deployed in numerous projects and organizations. Both messaging and service-orientation are also promoted by infrastructure offerings such as enterprise service buses (ESBs). However, there are various situations and needs, where the use of service-oriented approach instead of "traditional messaging" is justified:

- Message-oriented solutions define content, transmission infrastructure and invocation patterns using an established messaging style. Service-oriented solutions separate these concerns from each other and provide alternatives for different types of requirements.
- There are various alternative tools and methods available to support the specification and implementation of service-based solutions. Common development and integration tools and platforms support the implementation of services.
- Especially in situations where the interoperability needs focus on deterministic and functionally oriented requirements where service providers and consumers are easily identified, the identification and design of service-oriented solutions is straightforward and intuitive. However, both resource-oriented and activity-oriented services can be provided.

- Messaging infrastructure (MOM, message-oriented middleware) is widely used by service-oriented solutions for communication capabilities, but service-oriented infrastructures usually also provide support for service creation and hosting. Existing MOM implementations can be reused for communication in service-oriented solutions.
- A specific message-oriented transport may require modification of existing applications that communicate over different transports, or use of gateways or bridges, tying reliability, auditing etc. to the transport implementation. In fact, these policies apply across multiple transport channels, and should be supported by transport-agnostic service definitions.
- In message-centric solutions, the capabilities such as transformations, routing and process management are separate from the business services and deployed as part of the message broker or as separate server capabilities in the configuration. In service-oriented solutions, these capabilities can be defined and deployed as reusable services.
- Message-oriented programming model requires the developer to deal with entities such as queues, destinations, sessions, connections etc. Service-oriented programming model focuses on higher level of abstraction and coarse-grained business functions which hide the details of underlying infrastructure. This encourages top-down definition of business-centric services.

1.5 Service Definitions

1.5.1 Types of Services

There are many categorization schemes defined for types of services. For the purpose of this document, we will consider a simple classification scheme:

- **Business Services** – These provide specific business functionality, such as "Scheduling", "Order Management" and so on. These are often further subdivided into "Process Services" and "Core Business Services"[3]. The difference is subjective, but in general, core business services offer a set of operations, each of which performs one main single business action, whereas Process Services offer composite sets of processing that string several activities together (often calling underlying core business services at each stage)
- **Infrastructure (Technical) Services** – Infrastructure (Technical) Services – These are services provided to support the business services and are not specific to healthcare, but are often subject to specific requirements derived from regulation of healthcare information, for example by professional bodies or national legislatures. Examples include: Authentication, Authorization, Logging, Transformation.
- **Utility Services** – These are also supporting services, such as Printing, eMail, FAX etc.

[3] See CBDI Forum SOA Resource Center, www.cbdiforum.com, among others.

The Services that are in scope of this methodology are really the "Business Services" as described above, noting that "Process Services" are a way of dealing with some of the "delayed response" scenarios that are common in HL7 messaging, i.e. an initial synchronous acknowledgement followed by a subsequent asynchronous response. Note that these Services may be either "data oriented" (e.g. retrieve or update patient demographics) or "function oriented" (fill order or book appointment). Operations of both kinds may be combined within one Service definition where appropriate, although general best practices tend to keep them separate using a layered architecture.

1.5.2 Services, Specifications and Contracts

A key point that must be stressed is that the purpose in defining the Services is for interoperability, and as such the real purpose is to define the interface or "Service Contract", and not the internal implementation logic of the Service itself.

For the purposes of this document, the "Service Contract" is defined to be the formal specification of the Implemented Service, including technical level interface and operation descriptions, any Quality of Service limits or constraints and possibly terms of use / financial agreements. WSDL would be regarded as part of the Service Contract.

A Service Specification in this document is defined as analysis and design level documentation that describes the capabilities and functionality of a Service. This can include both logical (technology independent) and physical (technology specific) elements. The Service Specification would be an input used in creating the Service Contract. Note that the logical Service Specification is equivalent to the HSSP concept of the "Service Functional Model" (SFM). The template and process for this is fully defined elsewhere within HSSP.

In both cases, these only deal with "externally observable" behavior of the Service, and NOT the actual internal implementation. Where specification of the internal implementation is referenced, the term Software Architecture Document (SAD) will be used (borrowed from the Unified Process).

1.5.3 Structure of a Service Contract

Before describing the methodology, it is worth outlining the key structural elements within a Service definition, Specification or Contract. Typically within SOA, a Business Service definition consists of the following structural elements:

- **Business Service** – This is a set of cohesive business functionality that provides value added services to consumers of the service. Consists of 1 or more interfaces.
- **Interface** – A subset of a business service that group sets of operations of similar purpose.
- **Operation** – An individual atomic action within an Interface.
- **Message** – The data that is passed to and from the Service Consumer and Provider in the form of a document or set of parameters.

There are many alternatives for grouping operations into interfaces, which can be based on business logic groupings, technical factors or similarity of action. Often,

administrative operations are separated into an Administrative interface (e.g. to start and stop the service, to apply configurations etc.) For data oriented services, it is common to separate query operations, update operations and notification operations into separate Interfaces.

1.5.4 Interoperability using Web Services

Before continuing further, it is important to make a comment on interoperability. This part of the discussion is confined to web services specifically.

The current HL7 V3 Web Service Profile provides the useful capability to transport existing V3 messages using web service protocols. The intention here is for the service client to automatically be able to interoperate based on the messaging definition. The service definition becomes effectively superfluous.

The methodology in this document is intended to provide a "service based" approach, which means that the Service definition (or Service Contract) becomes key and needs to be available to the client at design time. Ideally, this would in the form of a fully approved industry standard specification. Where approved standards are not available, some kind of repository is needed for sharing service specifications at least between those providing and using the service. This repository is alluded to in Section 3.

Another aspect to consider is the recommendation to use human readable names for elements of the implementation level Service Description Language (WSDL) rather than the HL7 artifact IDs, since the Service Client will use the WSDL itself rather than trying to automatically derive what it may look like at a messaging level.

1.6 Target Audience

The intended audience for this document includes any organization or group planning on defining automated services within the healthcare domain. This could be standards development organizations (SDO), software vendors, healthcare payers or providers etc. The term "services definers" will be used throughout this document to identify these groups and individuals.

The two key targets are really HL7 domain committees that wish to define services, and also healthcare software vendors that are implementing services in their solutions. Those looking to fully understand the rationale and background for the approach are encouraged to read the document in its entirety. Those intending to define Services using the methodology should focus on Sections 4.3, 5 and 6. A future release may include a more concise "how to" guide if the overall approach is accepted.

Although this document is intended to provide a complete solution, various parts can be used stand alone, e.g. Section 5 (Design Guidance).

2 Methodology Requirements

Before describing the methodology, the key outline requirements were established, which provided direction as the methodology developed; these requirements also provide reviewers with assistance when analyzing the methodology.

2.1 Methodology Definition Context

1) Describe a method for deriving service definitions from existing HL7 V3 artifacts (at various levels – may include several alternate paths)
2) Give guidance on appropriate granularity for Services and Operations
3) Describe relationship to / fit with current HSSP Service Specification Development Framework

2.2 Methodology Content

1) Define services in terms of unambiguous, well-defined interfaces.
2) Describe services by their functional roles and responsibilities.
3) Define inputs and outputs of service operations and also the format and constraints on those inputs and outputs.
4) Describe service relationships in terms of messages and message exchange patterns. Where appropriate, relate to workflow / process.
5) Service interfaces must be defined independent of their implementation.
6) Support definition and implementation of self-contained services with clearly defined boundaries and service end-points to allow for service composability and interoperability.

3 High Level Process

3.1 Description

Figure 1 depicts an overall high-level process for producing Service Specifications (HSSP, SOA4HL7 and other models) in the form of a flowchart.

Appendix A includes a set of diagrams that place the SOA4HL7 methodology within the context of HSSP, basically “from HSSP looking in”. The diagram below presents a similar picture “from SOA4HL7 looking out”. The SOA4HL7 methodology itself is really only the dark blue shaded box. The light blue boxes represent relevant parts of the main SSF (reflected in Appendix A). The gray boxes are suggested additional steps that users of this methodology should consider.

Rather than describe each box in detail, the flow is described in narrative form below.

- Starting from a business need, where the scope and purpose for the required service have been defined, a check is made to see if the need is satisfied by an existing standard (be it HL7, HSSP or other) which can be used. Assuming there is no standard, a check is made to see if HL7 and/or HSSP have a relevant standard in progress that can meet the requirement. The service definer could then choose to participate and steer the standard accordingly.
- Assuming that there is no standard under development, a check should be made from any available sources to see whether an existing alternative specification exists (i.e. not a formal standard, but one that has been produced by going through this process previously, either by the same organization or other). Ideally, a cross-organizational repository of such service specifications would exist, but even within an organization this would be valuable.
- Again, if none is found, a check should then be made to see if the scope is covered by existing HL7 domain artifacts (static and/or dynamic). If so, then the SOA4HL7 methodology then comes into play and should be followed to produce the service specification. (Note that this could subsequently form the basis for a strong candidate for a future formal HSSP standard.)
- If the scope is not covered by an HL7 domain, then the service definer should look to see if other standard bodies have any useful artifacts that could be substituted. The SOA4HL7 could still then be followed (to a greater or lesser degree). Even if the domain models are completely custom or unique to a single organization, the methodology may still be able to be applied. However focus is placed on where existing HL7 model artifacts are found. Note - for a service to be considered "HL7 V3 compatible", the payload must be derived from the RIM and be able to be fully defined by a MIF.

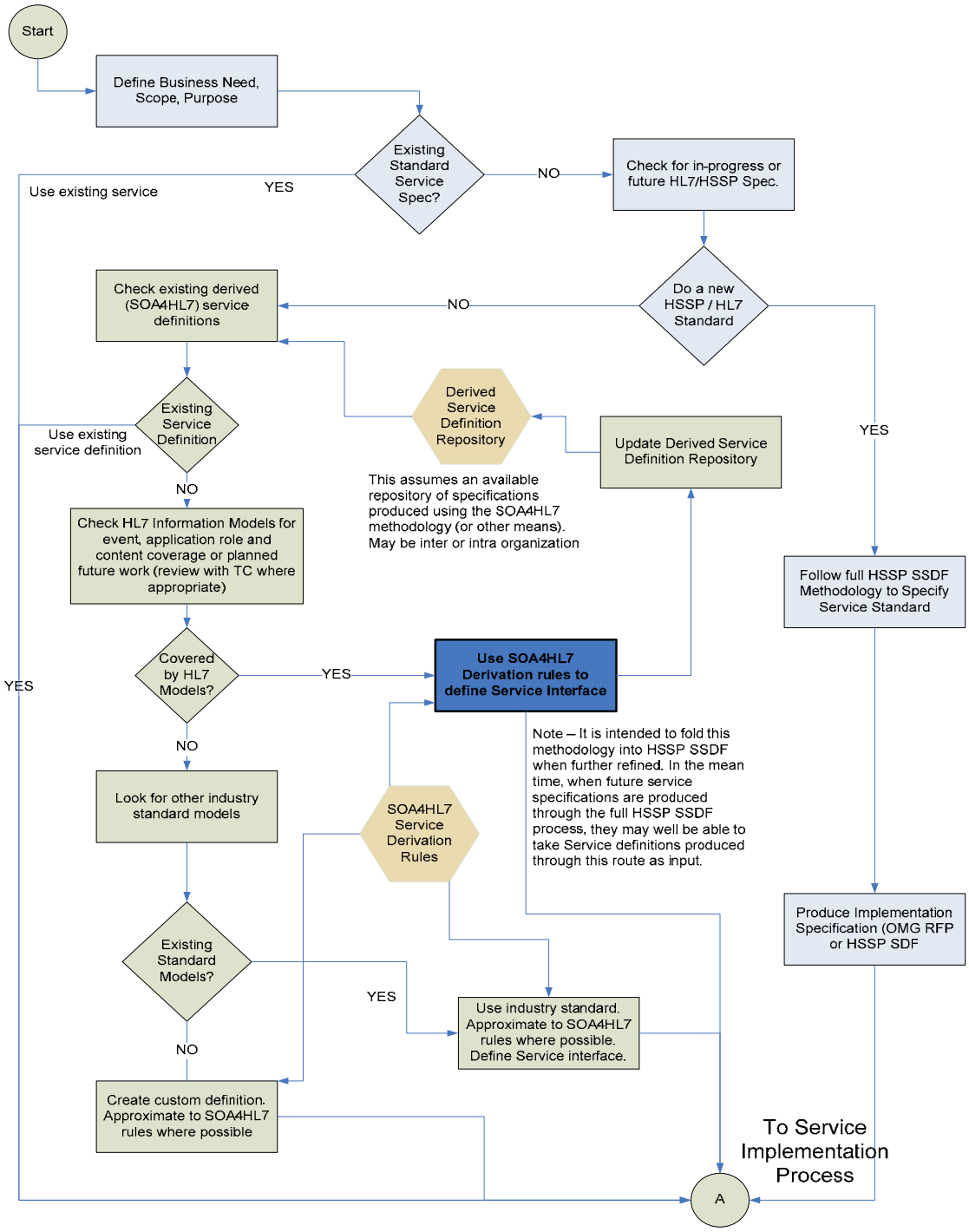


Figure 1: High-level process for defining service specifications

3.2 Other Context

There are some emerging methodologies and processes for service oriented modeling, analysis, design and development CBDI SOA Process, SOMA and Papazoglou. While this document does not presume any given approach, it is useful to identify end-to-end service-oriented process steps to relate this methodology to the overall SOA activities.

Service oriented modeling and architecture (SOMA) defines three service modeling steps: identification, specification and realization, with several sub-steps prescribing several artifacts to be delivered. Identification can start from domain decomposition and from analysis of existing systems, and include goal-service modeling to tie business goals to the identified service abstractions. During service specification, artifacts comprising SOA are formally defined. These include composite and atomic services, interfaces etc. Service model covers service invocation syntax and semantics, as well as service ownership, dependencies, versioning, and governance issues. Realization includes construction of actual services, processes and applications.

Papazoglou is another methodology for service-oriented design and development which includes planning phase and eight iterative phases

- **Planning**: business need, scope, purpose, initial requirements
- **Analysis**: business case analysis, alternatives for implementing business processes
- **Design**: identifying and specifying services and business processes in a stepwise manner
- **Construction**: development and description of service implementation, definition of technical interface description, also development of service consumers (clients)
- **Testing**: validate that requirements have been met and deliverables are in accordance with used conventions and conformance profiles
- **Provisioning**: definition of governance (central/distributed), certification, metering and rating, billing
- **Deployment**: roll out services and processes (or versions) to applications and users
- **Execution**: ensure that new process is carried out by all participants, services are found, static and dynamic bindings are operational, and messaging and interactions are operational
- **Monitoring**: measure and monitor service level, performance, availability etc., evaluate objectives.

Methodology in consecutive chapters of this document focus mainly on analysis/identification and design/specification steps of these processes, but have consequences to other parts of the process as well (e.g. construction, testing, deployment etc.).

4 Detailed Methodology for Service Definition

From the high level process detailed in Figure 1, this chapter concentrates mainly on the activity "Use SOA4HL7 derivation rules to define service interfaces."

While the overall aim of the SOA4HL7 work is to achieve a level of consistency, it must also be accepted that different process approaches will be taken in different circumstances. This section will focus on consistency of structure and definition, and identify a small number of options for deriving service definitions. The aim is to provide sufficient flexibility to meet most needs, balanced with sufficient consistency of method. The rationale is that a small number of well defined options are better than every organization going entirely its own way.

Service definition, design and development processes aim to identify the right services, organize them in a manageable hierarchy of composite services, and choreograph them together to support business processes.

This section will concentrate on defining the artifacts, while section 5 will set out guidance for making the appropriate trade-offs.

4.1 Approach Foundations (Top-Down vs. Bottom-Up)

The first level of discussion to consider is the basic philosophy of top-down vs bottom-up service definition.

- **Top-down**: Starts from detailed business requirements and process definitions. A top-down strategy starts with the requirements and business process models and refines them in a stepwise fashion down to a software implementation. This includes decomposition of the business domain into its functional areas and subsystems. In top-down development, business process models provide a blueprint for the identification of services. Services are then modeled and realized by service providers, and consumed by service consumers. In terms of HL7 artifacts, this would mean starting from the RIM, DIM, storyboards etc.
- **Bottom-up**: Starts from identified needs and existing solutions and applications. A bottom-up strategy originates from the technical basis and works upwards to the requirements and business process models by building services on a top of existing (legacy) systems. This includes analysis and utilization of APIs, transactions, and modules from legacy systems such as mainframe or ERP applications. This often requires componentization of the legacy systems. Bottom-up approaches often include two activities: add a layer of services on top of legacy systems using wrappers and adapters, and re-factoring legacy systems. In terms of HL7 artifacts, the equivalent would mean starting from Message schemas, CIM/LIM (or HMDs) or parts thereof.

A compromise is often suggested to these alternatives, which is termed “Meet-in-the-middle”, whereby both routes are taken for the same case and attempt to rationalize between the two.

4.2 Methodology Options

There are many different alternatives for the ways that existing HL7 artifacts could be used in defining service definitions. In order to provide some level of consistency without over-constraining service definers, based on the above discussion, two archetypal options are defined below. These are as follows:

1) Model Driven

Use the HL7 domain and dynamic models without specific message level constraints, i.e.

- RIM (Reference Information Model)
- Domain
- Topic
- DAM (Domain Analysis Models)
- DIM (Domain Information Models) / D-MIM (Domain Message Information Models)
- CIM (Constrained Information Model) / R-MIM (Refined Message Information Model)
- CMETs (Common Message Element Types).
- Vocabulary and Data Type definitions and restrictions
- Storyboard
- Use Cases
- Activity Diagrams
- Business Rules
- Application Roles (v2 or v3)
- Trigger Events (v2 or v3)

2) Message Driven

In addition to the elements listed above, this option would also make use of the actual message level constructs, i.e.

- Interaction
- LIM (Localized Information Model) / HMD (Hierarchical Message Descriptions)
- Message Type (v2 or v3)
- Message Segment (v2)
- Generated XML Schema

Note that as of this writing, the HDF itself is undergoing revision, so some of the V3 artifacts are changing. Where possible, both existing and new artifacts will be referenced. Note that in a typical "bottom up" approach, existing legacy systems would be a factor, but the purpose of this paper is to focus on how HL7 artifacts may be used.

Each approach listed above has advantages and disadvantages and will remain somewhat subjective; see Table 1 for a brief discussion.

Approach	Advantages	Disadvantages
Model driven	➢ Best business process alignment and "purest" SOA	➢ More analysis work ➢ Least reuse of existing HL7

Approach	Advantages	Disadvantages
	➢ Freedom to choose appropriate granularity and process/service division ➢ Consistency between different services ➢ Clear separation of concerns and responsibilities	artifacts ➢ Compatibility with existing applications requires extra work ➢ Additional effort required for local adaptation
Message Driven	➢ Quicker ➢ Accurate level of detail ➢ More reuse of existing artifacts ➢ Few changes required in applications	➢ Potential misalignment of services with business ➢ Inappropriate granularity of operations ➢ Expertise of the architecture or design of existing systems required ➢ Additional effort required for generalization

Table 1: Model Driven vs. Message Driven discussion

In general, given that the purpose is to try to preserve the main benefits of SOA, the favored approach would be more model driven. However, we must accept the reality that different organizations will favor different paths, and providing a means to achieve greater consistency will be valuable whichever route is taken.

In order to keep this approach simple, a single, compromise approach is proposed, which is basically model driven, but allows for using Message level concepts where it seems appropriate. To ensure that appropriate Operations are defined, it is important to work through the process from the higher level model as much as possible, and consider issues relating to process, reusability, granularity etc. before simply defining an operation to process each different message that is received.

4.3 Service Definition Methodology

The first sub-section (4.3.1) will identify the main elements of a Service that need to be defined and options for defining them, with a focus on using HL7 artifacts. This will be followed by a discussion of some generic process steps and where the artifacts fit (4.3.2). As discussed earlier, the latter is NOT intended to be a definitive guide on process. Finally, a section will describe specific considerations for creating WSDL definitions. Note that 4.3.1 and 4.3.2 describe some of the same steps but from a different point of view. The latter describes more of the overall process, following an MDA-style approach broken into three main sections, i.e. functional requirements and specification, then the Platform Independent Model then the Platform Specific Model. The former focuses specifically on elements of a service and how they may be derived, and does not differentiate the steps for producing logical business descriptions, PIM and PSM. However The process does provide cross-references back to the sections in 4.3.1.

4.3.1 Overview / Elements of a Service

Specification of the following elements will be described in this section: Service, Interface, Capability / Operation, Message (Input, Output, Exceptions)

Table 2 summarizes which HL7 artifacts would most commonly be used in defining the elements of a Service. The purpose is to provide a summary view of some of the in-depth analysis that follows.

Deliverable	HL7 v3 Artifacts that may be Used	HL7 v2 Artifacts that may be Used
Service	Domain, Topic, Storyboard, Application Role, Trigger Events	Chapter, Application Role, Trigger Events
Interface	Domain, Topic, Storyboard, Application Role, Trigger Events	Application Role, Trigger Events
Capability / Operation	DIM/D-MIM, Application Role, Storyboards, Activity Diagrams, Use Cases, Trigger Events (Interaction, CIM/R-MIM, LIM/ HMD, Message Type – if using message oriented level constructs)	Application Role, Trigger Events, (Message Types – if using message level constructs)
Message	RIM, DIM, CIM/R-MIM, CMETs, Vocabulary and Data Types (LIM, HMD, Message Type and Schema – if using actual message level constructs)	Message Types, Message Segments

Table 2: Commonly used HL7 artifacts for elements of a service

Depending on the approach, these elements could be defined in a number of steps. The end goal is to reach a technical level service definition. Ideally, this would be

produced first as a business level Service description, second as a Platform Independent Technology Specification and finally as a Platform Specific Specification (e.g. WSDL). The basic flow is depicted below.

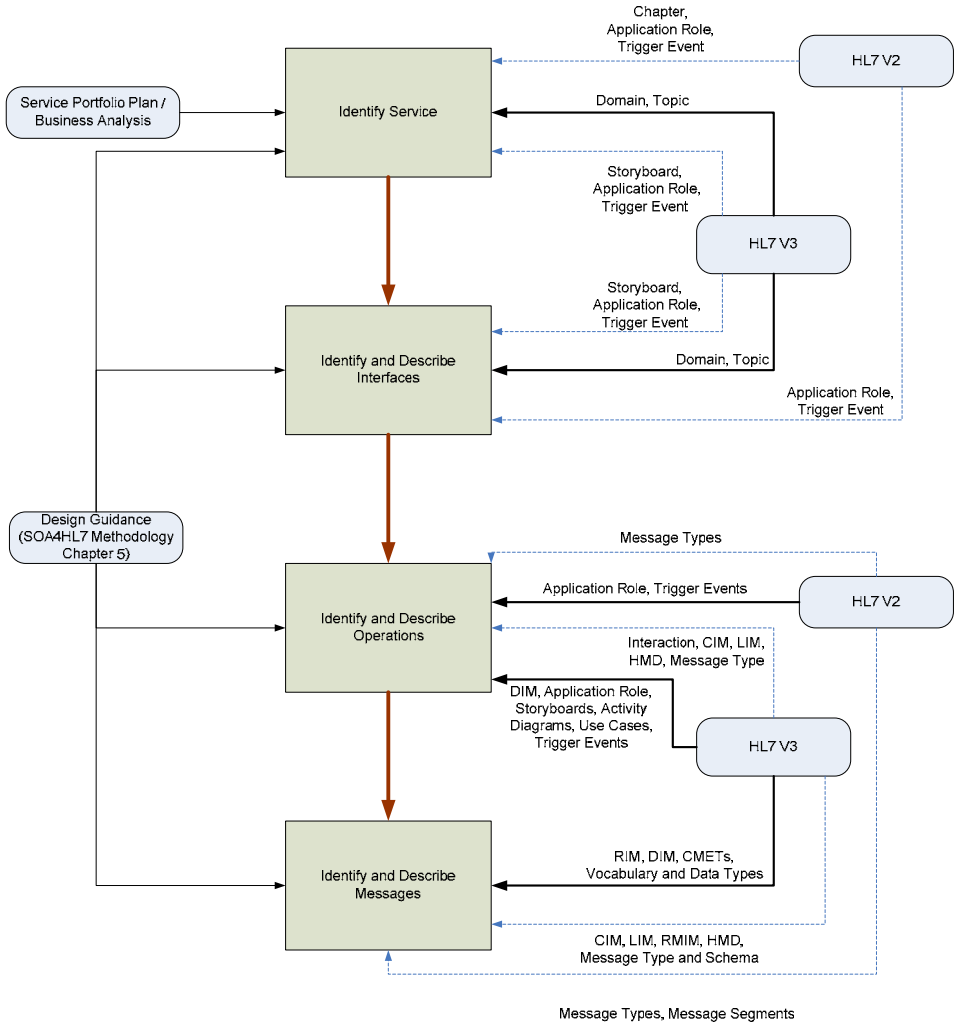
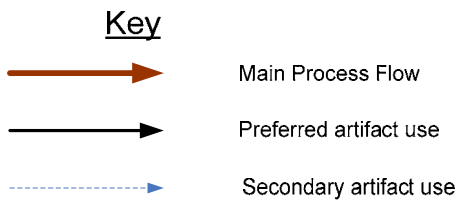


Figure 2 - Basic Process Flow



The process described in the next section is divided into these three stages. However, this section is aimed at showing how the main elements are derived, as far as possible - independent of the overall process, so are dealt with in a single section. Where there are key differences between a business and technical level specification, these are highlighted.

4.3.1.1 Identifying a Service

4.3.1.1.1 Guidelines

	General SOA	HL7 Artifact Based
Primary (preferred)	➢ Top down Service Portfolio Planning, based on analysis of business processes and business information. ➢ Service name usually includes the name of the "focal" business object where there is one followed by a "passive" verb. Names should be meaningful to business. ➢ Granularity based on cohesion and completeness (see Section 5 below) ➢ Use abstraction where appropriate. ➢ Examples: Member Registration, Flight Reservation, Employee Recruitment, Order Fulfillment etc.	➢ Use Whole Domains and/or Topics (where available) as basis, applying considerations of granularity and abstraction. ➢ Follow Service naming convention (see left hand column) ➢ Granularity based on cohesion and completeness (see Section 5 below) ➢ Use abstraction where topics and even domains are specific to one sub-type of a Domain model class (where a reusable service is viable) ➢ Examples: Eligibility Verification, Laboratory Order Management, Ambulatory Encounter Management (or Encounter Management), Scheduling.
Alternatives	➢ Based on Middle-in (Goal-Service Analysis) ➢ Based on existing legacy interfaces (bottom up) ➢ Based on "Meet-in-the-middle" combination of both top down and bottom up methods	➢ Aggregate a set of related Application Roles and or Trigger Events, applying granularity and abstraction criteria as mentioned above.

Table 3: Guidelines for Service identification

4.3.1.1.2 Rationale / Discussion

Ideally, identification of the service should be driven by business analysis. HL7 artifacts are structured into Domains and Topics that provide a good level for many services. One drawback of using "Topic" is that it is only a publishing concept rather than part of the formal model, but this does, in many cases, give a more appropriate result from an SOA perspective than other artifacts.

It is difficult to provide a precise answer to the question "What makes a good service?" However, the interfaces and operations included should all be closely related in a business sense and part of the same overall function with similar purpose and be fairly complete for that function. Granularity is also a concern, and will be fully covered in Section 5.

In some cases, the Domain level is more appropriate, in others the Topic or group of Topics is more appropriate. In the end, some subjective judgment must be used. Aggregation of several topics into a single service particularly applies where there are several topics for different subtypes of a main object, as in the case of "Person" and "Organization".

Service names are usually "passive" and usually include the name of the "focal" business object to which they relate, optionally followed by a passive verb. Note that this "object" may be data or function oriented.

In some cases, abstraction techniques can be applied to define a more reusable service. The Entity Identification and Retrieve, Locate Update Services (EIS and RLUS) currently in progress within HSSP are good examples of this re-use, where very generic interfaces can be defined which allow for specializations through a technique known as profiling. Another example could apply to Scheduling. As of this writing, HL7 V3 only contains Notifications of Appointments; although there is work in progress on Appointment Scheduling and a future plan to cover resource slots. An abstract service interface could be defined to schedule any kind of finite resource, which allows for specialization depending on the type of resource being scheduled. Another example could be Order Management (for various different kinds of orders). This approach can lead to very powerful and reusable services.

Ideally, the HL7 Domain Committee for each domain should identify the appropriate Services, although it should be noted that some services may well be across multiple domains, e.g. where a more generic or abstract service is defined as discussed above, e.g. Order Management. This would require cooperation between Domain Committees.

In terms of Business vs. Technical specifications, there should be little difference at this level, other than the possible restriction of one interface per service in certain technologies (see the Interface section 4.3.1.2).

4.3.1.1.3 HL7 Based Examples

The examples below are based on Domains and Topics. They are only meant to be illustrative, but should give a good idea of appropriate level and naming.

Suggested Service Name	Domain	Topics Covered	Comments
Eligibility Verification	Claims & Reimbursement	Eligibility Authorization	Eligibility and Authorization are closely related. The other Topics in the Domain relate to

Suggested Service Name	Domain	Topics Covered	Comments
			different kinds of business activity.
Laboratory Order Management	Laboratory	Filler Result Query	Could (should?) alternatively consider a more abstract "Order Management" Service. The Topics on their own are too low level and are closely related. Could use topics as separate interfaces.
Ambulatory Encounter Management	Patient Admin	Ambulatory Encounter	Could combine with In-patient Encounters into one "Encounter Management" Service, possibly with two interfaces. The word "Management" was added to make the purpose more clear. As a Domain, Patient Admin is too fragmented (not cohesive) to be a single service.
Scheduling	Scheduling + others (see comment)	Appointment + others (see comment)	Would could make this service more generic and include other scheduling over and above appointments. The Single Service would also include both Appointment Scheduling and Notifications. Resource slot and other scheduling would make sense as the same service interface, but may be different service implementation.

Table 4: HL7 based examples – Service Identification

4.3.1.2 Identifying Interfaces

Service definitions include 1 or more Interfaces. In many cases, Services may only have one "business" interface. For technical specifications, administrative interfaces may be defined for operations such as "Start Service", "Stop Service", "Pause Service" etc. The discussion below only deals with the "business" interfaces.

4.3.1.2.1 Guidelines

	General SOA	HL7 Artifact Based
Primary (preferred)	➢ If there is a natural split of business functionality into two or three areas, then define different interfaces. Questions of granularity are similar to that for the Service level. ➢ May split different interaction types/styles into different interfaces (particularly for data-oriented services), e.g. Query (read only) vs Update vs Notification (subscription based).	➢ As general SOA in left hand column. Also consider different Topics for interfaces if more than one in the Service. ➢ Examples Eligibility Verification (Service) -> Eligibility Query, Authorization (Interfaces) (Laboratory) Order Management (Service) -> (Laboratory) Oder Maintenance, (Laboratory) Order Query (Interfaces)
Alternatives	➢ Define a single business interface for the Service with the same name.	➢ Define a single business interface for the Service with the same name.

Table 5: Guidelines for identifying service interfaces

4.3.1.2.2 Rationale / Discussion

Even at a functional level, it is reasonable to wish to group read only, query style operations into one interface, updates into another, and possibly notifications into a third. These often have very different Quality of Service characteristics (performance, security, reliability etc.) and can be implemented and deployed as separate reusable elements.

At the technology specific or implementation level, this may actually be achieved simply by defining different Services. Issues of granularity and naming are similar to the Service level.

Where there is only one business interface for a service, it can be named the same as the Service. Where there is more than 1, this can be more directly related to a Topic, or without a relevant available Topic, use an aggregation of 1 or more Application Roles.

4.3.1.2.3 HL7 Based Examples

Further expanded examples are given below **(For illustrative purposes only):**

Suggested Service and Interface Names	Topics	Application Roles	Comments
Eligibility Verification: - **Eligibility Query** - **Authorization**	Eligibility Authorization	N/A	This could also validly be kept as a single Eligibility Verification interface, although splitting looks preferable given the different business purpose.
Laboratory Order Management - **Laboratory Order Maintenance** - **Laboratory Order Query**	Filler Result Query	N/A	Again, this could be kept as a single interface. Note also that some "Results" operations may not be separate operations, but asynchronous responses to requests to the "Filler". This can be handled by defining a "Process Service" that calls separate "Core Business Services" (Note earlier comment regarding an abstract Order Management Service)
Ambulatory Encounter Management - **Ambulatory Encounter Management**	Ambulatory Encounter	N/A	Only one interface. If the Service had combined Inpatient and Outpatient, then they could be separated at the Interface level.
Scheduling - **Scheduling** - **Notification**	Scheduling (V2) Appointment	N/A	See discussion above for the Service itself. Could provide abstract capabilities.

Table 6: HL7 based examples - Interfaces

4.3.1.3 Identifying Capabilities / Operations

4.3.1.3.1 Guidelines

	General SOA	HL7 Artifact Based
Primary (preferred)	➢ Identify individual business actions within the service scope, often based on steps in the business process, particular those that cross domain or system boundaries. ➢ For data-oriented services, ensure major business objects within scope have create, read, update and delete capabilities defined. ➢ For services where deterministic outcomes are required, use explicit, directed instructions rather than deriving implicit instructions from generic messages where possible. (see discussion below) ➢ Capability Names are active verbs, usually with business object as subject. Names should be meaningful business actions rather than system actions such as "update record". ➢ Defining Operations may also take specific interaction styles or performance implications into account, which may lead to splitting a Capability into multiple operations or occasionally aggregating Capabilities together. ➢ There are many approaches	➢ Identify individual actions from Storyboards, Use Cases, Activity Diagrams, Application Roles, Trigger Events ➢ DIM/D-MIM – look at major classes in scope and ensure that main create, read, update, delete actions are supported. This applies to services which manage and control data. ➢ Follow naming and granularity guidelines as for General SOA. Consider defining and/or using CMETs for data content of appropriate granularity operations ➢ Examples: Authorization (I/f) -> Request Authorization, Nullify Authorization Request, Query Authorization Request Status (Operations). Scheduling (I/f) -> Book Appointment, Cancel Appointment, Reschedule Appointment (Operations).

	General SOA	HL7 Artifact Based
	to defining queries, see the discussion below. ➢ Event-based or pub-sub issues will be considered in later versions of this document ➢ Granularity based on performance considerations and adaptability (see Section 5 below) ➢ Examples: Book Flight, Reserve Resource, Check Credit, Update Address, etc.	
Alternatives	➢ (Bottom-Up) Use existing legacy interface APIs or messages and re-define or wrap them as service operations.	➢ Use individual Interactions and/or CIM/R-MIM/LIM/HMD/Message Types and define operations for each

Table 7: Guidelines for identifying capabilities / operations

4.3.1.3.2 Rationale / Discussion

Business capabilities may be defined in a business level, non-technical model, then refined into Operations in a technical model. They represent the same basic concept. In some cases, a single capability may lead to more than one operation, or less frequently multiple capabilities in a single operation.

The Capability/Operation is the actual "unit of functionality" or the actual individual actions to be carried out. It is at this point that differences between Top Down and Bottom Up approaches really materialize. Appropriate actions can be based on the Storyboards and Activity Diagrams if available, as well as Application Roles and any identified Trigger Events. Section 5 contains guidance on appropriate granularity for Operations.

The need for a "deterministic outcome" must also be considered:

- Where this is not needed, i.e. in an event publishing type scenario where the publisher is notifying others that something has happened but is not instructing particular action, then an EDA (Publish and Subscribe) approach is ideal. In this mode, it is acceptable and common to use general-purpose messages and derive appropriate meaning from them.
- Where deterministic outcomes are required, use explicit, directed instructions or messages rather than deriving implicit instructions from generic messages. This keeps semantics clearer and better aligned with business.

In a messaging world, it is common practice to define large, multi-purpose messages and derive the appropriate instruction and meaning from them in "hidden" business logic. This obscures the behavior from the service client and often leads to non-deterministic states of objects. Wherever possible, explicit actions should be identified that take deterministic action. This does NOT imply that the "how" should be exposed, but it is important that the "what" is made very clear.

There is a trade-off, in the sense of preserving loose-coupling, but where there is a genuine need for dynamic logic, which does process generic messages, this should be separated into a process layer or service, which itself calls the more explicit services; or Publish and Subscribe architectures should be used.

One specific current example from HSSP is in the Entity Identification Service (EIS). EIS defines an explicit Capability/Operation to Create an Entity (Reference). Current messaging solutions often intercept different message types such as registration messages and implicitly derive the need to create an entity reference and create the reference all within one functional unit of processing. It is better architectural practice to separate these two steps so that the derivation of need is separated from the actual action. This increases modularity and reusability and better surfaces deterministic outcomes.

In cases where message level constructs are being considered, each Interaction within the scope of the Interface should be considered i.e. from the point of view of the Service Provider. Even for the "top-down" case, it may be reasonable to consider aggregating a number of interactions and messages together into a single operation, particularly when defining the data content.

Capabilities/Operations should be named actively, as in "action verb", usually followed by a noun (the name of the focal class.)

Another consideration is the interaction pattern or choreography. In current HL7 messaging scenarios, it is common practice to include a parameter to indicate the nature of the response e.g. immediate response vs. delayed response. In services, it is more typical to define one or more different operations where the behavior required is different; once again meaning, that the design time behavior is more deterministic.

The overall impact on the client is similar, in that instead of determining which value to set on a parameter, the client identifies a different operation to call. Where a specific sequence of events is required, e.g. an immediate acknowledgement followed by a later asynchronous response, an "orchestration" can be used which controls the overall choreography. This is not relevant to the definition of the service interface itself, although an orchestration can itself be exposed as a service.

In some cases, identification and definition of operations can be dependent on the data that is being exchanged. In a messaging world, it may be common to define a large message with many different sub-components within the message. Consideration should be given to identifying different operations that deal with specific subsets of the data if this is a common business scenario; this is subject to any guidance related to granularity. A simple example might be an operation to maintain a patient's address, as well as, an

operation to maintain demographics. With this in mind, CMET flavors are worth considering, i.e. consider creating new CMETs that fit with service operations. The following discussion considers a specific example of this:

Consider the X12 HIPAA Eligibility Verification Request Response Transaction, Some have proposed making this a "service" which works in a legacy system comprised of different files, accessible through several point-to-point interfaces or "screen scraping" for covered parties or contracted providers, both of which are associated with covered services and their various limitations related to diagnosis, gender, age, quantity, care setting and provider type limitations; associated by plan.

However, in a service oriented architecture, this transaction would need to be processed as a series of service calls to distinct modules which contain the data of record for the enterprise, e.g., member registry, benefit plan registry, provider registry. So business domain model design should reflect this decomposition in the way that the component models of the domain are chunked out.

For example, we have the ability to design domain model with discrete modules or Common Message Element Types (CMETs) which can be reused as query services to a registry, e.g., for patient demographics or carried as informational components in a composite service, e.g., a notification to subscribers that a particular patient is scheduled for a procedure. The CMETs can carry more or less information depending on how the type of coupling required by the architecture. For example, the CMET may simply carry the patient identifier because the other application either already has the other information about the patient, doesn't need the other information about the patient, or can retrieve the other information about the patient via a query to a shared patient registry. If domain models are designed with this flexibility in mind, then the use of CMETs would be increased, and an understanding about the benefits of keeping key information optional.

There are many approaches to defining query operations, key differences being those with fairly stable or fixed parameters vs more generic open queries with SQL-like or XQuery syntax. The HL7 "Query By Parameter" approach defines a generic structure that can be used to control sorting, limit the number of responses and identify parameters.

The common trade-offs of flexibility against performance occur in query design. Defining a fixed structure, or a set of related structures provides good performance at the expense of flexibility. One fairly common approach is to define a small number of queries which return increasing amounts of data, e.g. a basic, medium and full data set.

4.3.1.3.3 HL7 Based Examples

Examples shown below **(for illustrative purposes only):**

Service, Interface and Operation Names	HL7 Artifact (if any relevant)	Comments
Eligibility Verification: - Eligibility Query - **Query Eligibility** - Authorization - **Request Authorization** - **Request Immediate Authorization** - **Nullify Authorization Request** - **Query Authorization Request Status**	Eligibility Query Request (Trigger Event) Authorization Request (Trigger Event) Authorization Nullify Request (Trigger Event) Authorization Query Request (Trigger Event)	These operations fairly closely match the HL7 Messaging Interactions, but there would probably not be separate operations defined for different Specialties, although it could be done that way too. This would be handled by an input parameter indicating the request type. This leaves a closer match with the Trigger Events rather than the interactions. Note – defining separate operations is one way to deal with situations where the service requestor can indicate whether an immediate response is required. Nullify and Query would only apply in the asynchronous response case.
Laboratory Orders - Laboratory Orders Management - **Order Lab Observation** - **Cancel Order** - **Etc.** - Lab Order Query - **View Order**	Order Activate (Trigger Event) Order Cancel, Order Nullify (Trigger Events) Find Order, Find Result, Find Promise (Trigger Events)	Correct mapping would take more in depth analysis.

Service, Interface and Operation Names	HL7 Artifact (if any relevant)	Comments
Ambulatory Encounter Management - Ambulatory Encounter Management - **Create Ambulatory Encounter** - **Update Ambulatory Encounter** - **Close Ambulatory Encounter**	Ambulatory Encounter Started (Interaction) Ambulatory Encounter Revised / (Interaction) Ambulatory Encounter Ended / Aborted / Nullified (Interactions)	In V3 documentation, appointment notifications are also included, these appear to overlap with scheduling so have been omitted. Not clear without further analysis whether separate operations would be used for abort / nullify etc. as opposed to a single close operation.
Scheduling - Scheduling - **Request Appointment** - **Cancel Appointment** - **Reschedule Appointment** - **Check Available Appointments** - **Etc.**	Event S01 (V2) . Event S04 (V2) Event S02 (V2) Event S25 (V2)	Other than Notifications, these are not yet defined in V3. There are V2 events that correspond to some of these operations, although there may be some difference in granularity. However, it would also be fairly easy to map some of these operations to the resulting notification trigger events.

Table 8: HL7 based examples - Interface Capabilities

4.3.1.3.4 Define Exceptions

Identify all business level exceptions for each Operation explicitly in the service description. HL7 V3 provides a standard set of metadata for errors, which in the case of web services maps fairly closely to the SOAP fault structure. When using WSDL, it is common practices to define a single Fault element with a code structure to define the actual errors. In HL7 cases, the standard metadata structure should be used (as defined by the *AcknowledgementDetail* class. Any existing identification of error conditions in HL7 documentation (or elsewhere) can be used as a source.

4.3.1.4 Identifying Message Content (Capability/Operation Input and Output)

4.3.1.4.1 Guidelines

Define the information content consumed and produced by each Capability/Operation.

	General SOA	HL7 Artifact Based
Primary (preferred)	➢ For data oriented services, this is normally complete business objects or sets of related objects ➢ For functional services, the appropriate parameters and return values are defined. ➢ It is important to consider extensibility, and use of more general document type approaches are now more common. ➢ Messages are normally named as a (qualified) noun describing the business content. ➢ Examples: Flight Reservation Details, Credit Card Verification Request, Reservation Confirmation.	➢ If there is a matching CIM/R-MIM for the scope of the operation then this should be used. Otherwise start with DIM and identify appropriate classes in scope. ➢ Look for relevant reusable information structures (CMETs) ➢ May use aggregation of 1 or more CIMs / R-MIMs, LIMs, HMDs or message content. ➢ Data Types and Vocabulary should be based on existing V3 artifacts where they exist. ➢ Examples: Patient Demographics, Laboratory Order, Eligibility Authorization Request, Appointment Confirmation
Alternatives	➢ Bottom Up – Existing API content or message schema.	➢ Use previously generated Message Schema

Table 9: Guidelines for identifying message content

4.3.1.4.2 Rationale / Discussion

At the business capability level, this should be as an information model (preferably UML).

For the Operations, at the PIM level this can still be represented as a UML model or other neutral form that represents the data hierarchically (as in the HL7 CIM/R-MIM, LIM or HMD).

At the PSM/implementation level, assuming an XML solution, an XML Schema should be produced. The rules for defining or generating XML Schema are beyond the scope of this document, but a suggestive worked example is included below.

Where the Operation was derived directly from an Interaction, and also in some other cases, there may already be an existing HL7 V3 R-MIM available which **closely** matches the requirements and appropriate granularity of message. If this is the case, then it should be used. If the granularity does not seem appropriate, or if there is no R-MIM available, then either a more constrained model may be needed (to get more fine grained) or revert to a higher level model (e.g. DIM). A search should also be made for relevant CMETs. Vocabulary and Data Types from HL7 should be used to define the actual individual data items where a fully defined more coarse grained structure is not available.

As an alternative, where they already exist and are relevant, existing standard (non-normative in the case of HL7) XML schema or schema fragments may be reused.

Messages should be named using a noun describing the business content. At the implementation level, it is typical to append “request” or “response” to the message name to indicate its role in an operation.

Notes:

- See also http://www.hl7.org/v3ballot/html/help/hdf/hdf.htm#HDFAnnex2, which discusses the use of UML with stereotypes to produce Domain Analysis models aligned with the RIM.
- For a service message to be considered "HL7 V3 compatible", the payload must be derived from the RIM and be able to be fully defined by a MIF.
- In the overall HSSP process, Semantic Profiles explicitly define a way to take a group of related messages, possibly derived from RMIM, and relate them specifically to a Service Structure. Semantic Signifiers are the conceptual equivalent of PSM/Implementation level semantic constructs.

4.3.1.4.3 HL7 Based Examples

Examples shown below (**for illustrative purposes only**):

Service, Interface, Operation and Message Names	HL7 Artifact (if any relevant)	Comments
Query Eligibility ➢ **Eligibility Query Request** ➢ **Eligibility Query Response** Request Authorization	Eligibility Event Query Request (RMIM) Eligibility Event Query Results (RMIM)	In these cases, there seems to be reasonable correspondence between existing messages and operation content. Further analysis would be needed with respect to granularity and other aspects before determining

Service, Interface, Operation and Message Names	HL7 Artifact (if any relevant)	Comments
➢ **Authorization Request** ➢ **Authorization Response**	Authorization Event Query Request (RMIM) Authorization Event Complete (RMIM)	appropriate content.
Request Appointment ➢ **Appointment Request** ➢ **Appointment Response** Cancel Appointment ➢ **Appointment Cancel Request** ➢ **Appointment Cancel Response** Reschedule Appointment ➢ **Appointment Reschedule Request** ➢ **Appointment Reschedule Response** etc.	V2 Messaging segments (ARQ, SCH, RGS, AIS, SIG, SIL, SIP, ARP) could be used as a basis or relevant parts of Notification models from V3.	Other than Notifications, these are not yet defined in V3. There are V2 messages that correspond to some of these, although there may be some difference in granularity. However, it would also be fairly easy to map some of these to the resulting notification messages.
Lab Order ➢ **Lab Order Request** ➢ **Lab Order Response**	Placer Order (CMET) Fulfiller Order (RMIM)	

Table 10: HL7 based examples

4.3.2 Activities / Process Steps

This section identifies typical steps carries out within a Service Development process, and where the use of HL7 artifacts may fit in. How these artifacts are used in deriving specific elements of the Service definition was covered in the previous section. This section also considers a wider view in terms of planning and developing overall architecture solutions and not just the service interface definitions (which is the primary aim of this document). This includes the internal service implementation logic, and also usage patterns and process implementations. However, it is believed that this additional material provides useful context. A cross-reference is included below back to the main steps identified in the previous section.

- Where an activity is beyond the scope of solely interface definition (i.e. mainly concerned with the internal logic or external process / choreography issues), the text is *italicized*.
- References to the deliverables from the previous section are in **bold type**.

4.3.2.1 Requirements and Functional Specification

This stage produces the business level definition of the service.

Step	Description	HL7 Artifacts
1) Define requirements	Identify the set of requirements to be included in the service, the basic integration model and relevant process. Document the interoperability scenarios/use cases. For example, order fulfillment, observation result notification, or person identity lookup. Top down - process models, business objectives, measurable goals, categorization and decomposition of the business environment into business areas and business processes etc. Bottom up - identify functional features to be reused.	Use Cases, Storyboards, Application Roles, Trigger Events, Interactions
2) Describe capabilities (process and information)	**This step and step 3 covers steps 4.3.1.1 and 4.3.1.2 above, i.e. identification and description of Services and Interfaces.** Top down – Functional and Information models, definition of process scope (where the process starts and ends, related users and stakeholders, inputs and outputs for each of them, different types of events and activities, conditions and synchronization).	RIM, DIM/D-MIM, Use Cases, Storyboards, Application Roles, Business Rules, Trigger Events, Interactions, CIM/R-MIM, LIM, HMD, Message Types

Step	Description	HL7 Artifacts
	Identify the Business Objects that are relevant to interoperability. For example, "LaboratoryOrder", "LaboratoryObservation", "Person", "Patient" may be either exposed by the Service or used by the Service. For instance, Lab may expose "Order" and "Laboratory Observation" but use the 'Person" or "Patient" identity information provided by the Person Service. This analysis will also help identify potential dependencies on other Services. The Business Object definitions should already be represented in Domain Models. Review the Domain Models looking for those classes of interest. If the object don't appear at all or they are incomplete, ideally the Domain Models should be updated. For example the "Laboratory Order" must appear as a specialization of a generic "Order" or needs additional components. Once the business objects are identified, the behavior should be described based on the state transitions in scope. Similarly, any notifications triggered by those state transitions must be also identified. In order to determine any behavior required from other services, identify dependencies on interfaces and notifications. Bottom up – Service registry, portfolio of available services and processes, legacy systems for wrapping.	
3) Identify and name service components	Identify Service Provider, Service Consumer. **Finalize Service and Interface Names and descriptions.** Define responsibilities of service provider and also consumer in relation to the identified interfaces and capabilities.	Domain, Topic, Application Role, Trigger Events
4) Map requirements to components	**This is the beginning of step 4.3.1.3 (completed in step 4 of the PIM below)** Map the identified requirements to the responsibilities and interfaces of the identified components. Fully describe the capabilities in business terms (not as formal "operations"). Define features as either required or optional.	DIM/D-MIM, Application Role, Storyboards, Activity Diagrams, Use Cases, Trigger Events, Interaction, CIM/R-MIM,

Step	Description	HL7 Artifacts
	Define extensible features and mechanisms for extensions.	LIM, HMD, Message Type
5) Produce logical service specification	Produce a logical Service Specification. This pulls together the business context and requirements and functional descriptions into a complete logical description of the Service capabilities. The HSSP SFM Template may be used for this document.	N/A

Table 11: Requirements and Functional Specification

At this point, the equivalent stage of the "SFM" in the main SSDF methodology has been reached, i.e. we have a functional specification.

4.3.2.2 Solution Specification (PIM)

This stage defines the initial technology solution, but still at a platform independent level.

Step	Description	HL7 Artifact
1) Refine interaction solution	Refine the interaction solution, for example the deployment and interaction style. Consider centralization vs federation, interaction patterns	N/A
2) Refine component definitions	Identify integration points in the architecture of the participating systems. Refine the responsibilities of the components, identify possible extension needs and needed security features. *Define internal logic specification, and/or how legacy system logic will be used.*	N/A
3) Define detailed dynamic model	Specify interaction sequences, which may also contain user interaction.	Storyboards, Activity Diagrams, Use Cases, Trigger Events, Application roles, Interactions,
4) Specify operations and messages	**Completion of step 4.3.1.3 and 4.3.1.4 above. Operations** and Information contents and semantics (**messages**) are specified as e.g. document definitions in document style, parameter definitions in procedural style, and	Storyboards, Activity Diagrams, Use Cases, Trigger Events, Application roles,

Step	Description	HL7 Artifact
	functional needs as e.g. operation names and document/message types. The interface details should be unambiguous, well-defined interfaces (inputs and outputs of service operations + their functional constraints and generic format) Define features as either required or optional. Refine extensible features and mechanisms for extensions. All business level exceptions should be identified and described for each operation. *May also define further interactions that are part of the service implementation, e.g. interactions with legacy systems behind service facades.*	RIM, DIM, CMETs, Interactions, CIM/R-MIM, LIM, HMDs, Message Types / Definitions, Vocabulary and Data Types
5) Define QoS / implementation considerations	Refine the requirements for implementations or further technical and policy specifications. QoS - Policy definitions concerning security, performance, reliability, scalability, availability, transactional requirements, change management and notification etc.	N/A
6) Produce Platform Independent Model / Specification	Produce a Platform Independent Model / Technical Specification. This provides a detailed level platform independent representation of the service functionality.	

Table 12: Solution Specification PIM Steps

In the main HSSP process, this is part of the RFP submission process. In general, submitters would be asked to include a Platform Independent Model for their solution.

4.3.2.3 Technical Specification (PSM)

Step	Description	HL7 Artifact
1) Define implement-ation scope	Select and group features from functional specification to be implemented, if not all are implemented or mandatory	N/A
2) Technology selection	Refine the required technical capabilities of the solution and link them to available	Follow SOA4HL7 Architecture

Step	Description	HL7 Artifact
	technologies, including necessary routing, protocol mediation and other transformation mechanisms. Consider the suitable technology and interfacing options of participating systems or existing solutions which are to be used. Select set of technologies to support the service (transport (messaging, enveloping, reliability etc.), interface (functionality, information), security.	guidelines and/or full HSSP OMG RFP process.
3) Produce Platform Specific Model (PSM)	Refine functional specification with technology-specific features (e.g. simple or complex types, messaging style such as data-oriented or rpc or process-oriented)	
4) Define environment services	Identify technology-specific services (/consumers), interfaces, operations and parameters; specify their responsibilities	N/A
5) Produce technology specific interface specification	Create technology-specific interface specification: e.g. for web services: 1. describe **service interface** 2. specify **operations and messages, including exceptions** 3. designate messaging (e.g. SOAP) and transport (e.g. HTTP) protocol 4. define bindings and actual service location 5. If applicable, publish to registry	
6) Define Conformance	Define technical conformance levels	
7) Produce Release Document-ation	Document implementation-specific features of the service, infrastructure etc, extensibility options etc.	

Table 13: Technical Specification PSM Steps

Step 5 (and to some extent 6 and 7) would include the final output from the full RFP submissions in the HSSP process.

Note on generation of physical models: Approaches such as OMGs Model-Driven Architecture (MDA) and in tooling are beginning to allow the possibility of generating PSMs from detailed models. However, PSM generation is still not uniform but tool- or project-dependent at the moment. There is no agreed abstraction or functionality level especially regarding what goes in CIMs and PIMs. Most of the SOA literature does not see "generation from higher level models" among main service acquisition options to date

4.3.2.4 Implementation and Deployment

The focus of this document is on defining Services. However the services have to be implemented and deployed. This will include acquisition or development, testing, deploying and support. This should also involve registering the services in a services registry. The accompanying architecture document discusses some of these issues.

4.3.3 WSDL Specifications

This section specifically considers creation of WSDL specifications in light of the discussion above. This is based on WSDL v1.1. (The emerging WSDL V2.0 standard will be covered in a later version) and will include guidance on style and naming. This only covers the "logical" section of the WSDL and does not cover any of the technology specific binding information. This will either be considered in the Architecture document or a later version of this document. This covers the Port, Binding and Service elements. Also, see the note in Section 1.5.4 with regard to WSDL naming.

Also note that the definitions should conform to any current WS-I profiles currently in existence. Any WS-I rules should override anything in this document.

WSDL <DOCUMENTATION> element.

- Recommend briefly documenting purpose, scope, policies, non-functional characteristics etc.

  This is optional but strongly recommended. This information should be fully documented in a design level specification, but it is useful for clients to be able to retrieve all key information from one document.

WSDL <PORT TYPE> element.

This is an abstract (logical) definition of an interface.

- Create one for each "Interface" as identified in Section 4 above
- Name the Port Type "XxxXxxXxx" where XxxXxxXxx = the business descriptive Interface Name as a "UpperCamelCase" string

WSDL <OPERATION> element.

- Create one for each "Operation" as identified in Section 4 above.
- Name the Operation "XxxXxxXxx" where XxxXxxXxx = the descriptive Operation Name as a "UpperCamelCase" string

WSDL <INPUT> element.

This identifies a message that the operation accepts as input

- Create one for the input message for the "Operation" as identified in Section 4 above.
- Name the Message "XxxXxxXxx" where XxxXxxXxx = the descriptive Message Name as a "UpperCamelCase" string

WSDL <OUTPUT> element.

This (optionally) identifies a message that the operation responds with as output.

- Create one for the output message (if any) for the "Operation" as identified in Section 4 above.
- Name the Message "XxxXxxXxx" where XxxXxxXxx = the descriptive Message Name as a "UpperCamelCase" string

WSDL <FAULT> element.

- If identifying errors, create one Fault message element for the “Operation”.
- Name the Message “XxxXxxXxxFault” where XxxXxxXxx = the operation name as a “UpperCamelCase” string.

WSDL <TYPES> element.

This must include an explicit definition of the XML Schemas used in messages. No specific guidance on schema design or naming is included at this stage. May include in a later version

WSDL <MESSAGE> element.

Each message used in an Operation must be declared as a MESSAGE element. Naming is described under the various Operation components described above.

5 Guidance for Design Decisions

This section provides guidance on appropriate granularity and other central attributes for services and operations. In addition to functionality and information, structural and external software quality attributes are considered. Structural software attributes include coupling and cohesion, and main external software quality attributes (ISO/IEC 9126-1:2001) include functionality, reliability, efficiency, usability, maintainability, and portability.

In SOA design, solutions for a given process are seen as composite federations of services connected via well-specified contracts. These business services or processes can be composed of finer-grained services that are supported by infrastructure and management services such as those providing technical utilities such as logging, security, or authentication, and those that manage resources.

This section also identifies some of the trade-offs and approaches that should be considered when designing a new service. These acknowledged software engineering best practices lead to solutions that are easier to maintain and more responsive to ongoing change to business level requirements and processes.

One other question that needs to be considered, but is not covered in this version of the document is: Are there optimal design patterns for the development of HL7 domain artifacts, e.g., CIMs/R-MIMs and Messages or CDA to better support their use as service payloads? For example, to maximize service reuse, should the HL7 Development Framework provide guidance on designing cohesive models at the appropriate level of granularity to better enable their composition and orchestration as service components?

5.1 Service Design Considerations

5.1.1 Modular Design

Modular design refers to the decomposition of a software system into a series of units that are loosely bound to one another and which implement a set of features that are closely related. Modular systems are easier to maintain and are less fragile since changes to such systems can be typically isolated to a set of specific areas without impact to other portions of the overall system. Successful modular design relies on definition of simple and abstract interfaces between functional units; interfaces which adequately encapsulate the internal implementation details of the unit, allowing these details to be changed without affect to other units within the system.

5.1.2 Tolerance of Independent Invention

Effective systems are not only modular, but they should also be flexible, allowing individual units to be provided from a wide variety of sources and allowing the system as a whole or in part to be used as a component of larger systems. Systems that are not flexible are difficult to integrate into larger solutions and are difficult to upgrade in the event superior implementation of individual system units are realized. Units within a

flexible system are specialized, focusing on doing one thing well while leaving other tasks to other units within the system. A unit that is too broad in its functional mission is difficult to reuse within subsequent, unanticipated use cases and may contain valuable function that cannot be reused since it is not packaged as a standalone, modular function.

5.1.3 Types of main functional requirements

A key consideration is the nature of the main functional requirements, which leads to different "types" of services. Common types are:

- Data Oriented / Information sharing: based around creation, update and retrieval of messages and/or documents
- Function Oriented / Shared functionality: based around reuse of business functionality
- User facing / usability: based around providing direct user functionality, such as portals, context management services, WSRP portlets. These can provide fine grained operations as opposed to the more coarse grained operations for application to application interactions. Note that many service implementations may also include finer grained local operations that are "private", i.e. used by the external facing "public" operations.
- Process Oriented / workflow management: based on coordinating invocation of a number of services into a sequence or choreography, using orchestration or composition approaches.

One other kind of service that should be touched on is one that is "event based". The approach known as "Event-Driven Architecture" (EDA) is often paired with SOA in an overall solution (some see it as part of SOA, probably incorrectly). From a technology perspective, this is usually implemented as a "Publish and Subscribe" solution. EDA is not covered fully in the first iteration of SOA4HL7 work.

5.1.4 Adaptability

Ideally, design of a service and its operations should consider approaches that would enable the service to adapt to use within future systems and to deal with future requirements and the likelihood that the operating environment surrounding the service will be subject to change. This requires analysis of the service and identification of areas most subject to future change.

Usage context, information model, functionality, interaction with other components and method of communication are some of the areas which may undergo change when adapting a service for use within a subsequent solution. While a highly adaptable service is most desirable in the general case, one should also consider the impact adaptability has on ease of use of the service. In some cases, a pre-defined and configured service may be preferable; one that supports a plug and play deployment model. This is often the case in relatively closed systems where little evolution is expected and the resulting solution is expected to be stable and in use for a long period of time.

However, in more evolutionary situations, where the information a service deals with and the users and uses of the service are continually changing, it is often wise to build in a degree of configurability to a service to cope with the ever changing landscape in which it will be used.

5.1.5 Granularity

The level of granularity chosen for a given service and its operations is an important consideration since it can affect the degree of coupling and cohesiveness and responsiveness of the service and system it's used within. Service granularity refers to the scope of functionality and purpose of a service. Operation granularity refers to the functional scope and corresponding message size for single transactions. The following should be considered when defining the granularity of a new service and its operations:

- Fine-grained services in a bottom-up development model typically increase cohesion and decrease complexity of the service and coupling within the service. However, fine-grained services increase the number of connections and thus the coupling between collaborating services.
- Encapsulating legacy systems with a coarse-grained services layer will typically result in less of a development effort since it is easier to generalize existing functionality into coarse-grained service interfaces. This benefit is magnified when using an approach related to standard-based service definitions.
- Since coarse-grained services require less communication than fine-grained services, network performance is typically better in the coarse-grained case.
- Coarse-grained services are typically more easily identified by domain experts since they tend to map more directly to business process level activities.
- Coarse-grained services introduce increased coupling within the service, but increased cohesion from the viewpoint of the service user.
- When interactions involve transfer of accountability (typically found in B2B situations), self-contained interactions should be emphasized, leading to coarser-grained service operations.
- Coarse-grained services are often used for inter-enterprise communication and for intra-enterprise communication between business applications due to their focus on activities defined at the business process level.
- Finer-grained services are best used when communicating between parts of a composite application inside a given organization where the internal network is faster and more stable. High level of interaction between fine-grained components can result in unacceptable overhead when used in a distributed, cross-enterprise environment.

5.1.6 Abstraction level and composition

In SOA design, solutions for a given process are seen as composite collaborations of services connected via well-specified contracts. A business or process service itself can be composed of finer-grained services that are in turn supported by infrastructure and management services such as those providing technical utility such as logging, security, or authentication, and those that manage resources.

One common method of abstraction in SOA is to provide more generic functional capabilities as operations. Examples of this are the current HSSP Specifications for Entity Identification (EIS) and Retrieve, Locate, Update (RLUS). These both can deal with many different kinds of content.

Similarly, one can look at management of Pharmacy, Laboratory, Radiology orders and see functional similarities. Defining an Order Management service from a functional perspective that can be specialized for Lab, Pharmacy and so on has potential merit.

5.1.7 Cohesion/coupling/complexity

5.1.7.1 Coupling

Coupling is a measure of the degree to which components rely on the inner workings of other components in a particular solution, necessitating re-engineering the whole solution when a piece or component changes. Loose coupling between individual services is desirable since it minimizes the overall system impact when a given service is changed or replaced. Loose coupling can be achieved by reducing the number of dependencies between services, eliminating unnecessary relationships and minimizing reliance on external services and specific infrastructure features. Flexibility of solutions and reduced design-time coupling are pursued by postponing different bindings to late phases in the systems development. These bindings include location (where the service is), interface (what the syntax of the interface operations and data elements is), data (what the data contents used through the service are) and semantics (the meaning of interface elements and operations) level.

There are a number of design-time trade-offs which can influence the degree of coupling between a service and its consumers. The following table illustrates several examples of service design trade-offs which can affect level of coupling:

Tighter Coupling	**Looser Coupling**
Use of identifiers and references when interacting with a service. Both service and consumer must understand how to acquire and interpret these identifiers and references.	Including all necessary information on a given concept within transactions between service and consumer. This may lead to some degree of redundancy and verbosity of information exchanged.
Reliance on a single, fixed, service instance. Disruption of this instance can have immediate impact on service consumers.	Multiple deployed instances of a service and virtualization of service location can lead to more resilient systems which can tolerate outages of a given service instance.
Strongly typed and strictly enforced service interface parameters. This combination can make a service interface more difficult to	Designated schemes to locally extend the data exchanged between service and consumer can be used to evolve a given

Tighter Coupling	Looser Coupling
evolve.	service interface while maintaining a degree of backward compatibility.
Reliance on specific external services or specific infrastructure features increases the number of dependencies which must be agreed on between service and consumer.	Self-contained services with minimal infrastructure dependencies result in fewer dependencies spanning service and consumer and a looser degree of coupling.
Use of small messages or fine-grained and numerous operations. Consumer must understand the set of operations required to perform a higher level business process activity and must understand order dependencies between these operations.	Use of larger-grained messages and/or documents related to business process state or business process level activity.
Synchronized service operations which rely on availability of the service. Request/reply operations where the consumer is blocked until a response is received results in tighter coupling; outage of the service will impact the consumer.	Asynchronous and/or guaranteed message delivery model leads to more loosely coupled systems that are more tolerant of periodic component outages.
Stateful operations where service maintains state based on previous consumer interactions. Management of state increases the number of dependencies between service and consumer.	Stateless operations or those in which prior state is transferred as part of each message.

Table 14: Coupling Definitions

5.1.7.2 Cohesion

The extent to which elements of a service or a solution contribute to one and only one task (functional cohesion), the extent to which activities and services use the same messages (communicational cohesion) and the extent to which services perform logically similar functions (logical cohesion). Highly cohesive modules can be used for a number of both intended and unanticipated purposes without dragging along a lot of functionality that is not germane to the central task at hand.

5.1.8 Completeness

When defining the operations for a service, particularly when defining a reusable standard, consider the “completeness” of the operations. For example, in data oriented services, are operations included that enable all appropriate state changes defined for the information object, (create, update, delete, retrieve, suspend, reinstate etc)

5.1.9 Design for Reuse

Service reuse is another important consideration when designing a new service. Typically, it is easier to reuse finer-grained, simple services across a broader set of solutions and use cases. While service reuse can lead to increased ROI, reusable services typically require greater development effort given the broader set of use cases they are designed to support. The process of designing reusable services tends to focus on making the service and its interfaces more generic and abstract. Reusable, generic services may also be derived by factoring common function and behavior out of an existing set of related concrete services.

5.2 Security

SOA enables loosely coupled applications to be assembled from a set of internal and external services (web services) that are distributed over a connected infrastructure. Each partner in the collaborating service must protect their sensitive data. In some cases, the partners must protect even the existence of a service from unauthorized probing. Finally, the partners must be able to enforce their collaborating transactions. Thus, an SOA must address issues of authentication, access control, encryption, non-repudiation, and authorization.

The distributed nature of SOA makes addressing security concerns a critical success factor. The primary concern in SOA is to establish an interoperable framework that enables security for services, applications, and users in a trusted environment and complies with established corporate policies. These standards and techniques to provide security in a SOA are evolving rapidly.

Note – overall issues of security are orthogonal to the process of designing appropriate business service interfaces.

5.3 Process Management

Some services, especially those that are coarser grained, include some degree of workflow or business process orchestration. This is often the case when a service is composed of a number of lower level services that are accessed in a prescribed sequence to implement a given business process activity. While it is possible to imbed this orchestration task within the service implement, it is often desirable to design the service to separate process orchestration handling from individual functional responsibilities. This can lead to a more agile service that can quickly react to changes in business process or changes in the underlying services used since many of these changes can be made at the workflow level without impact to the underlying functional units which together make up the aggregate function provided by the service.

5.4 Technical Governance

With SOA, you can expect that business process cycles will be different from vendor product cycles. As a result, it is inevitable that, in the case of long-running or long-lived processes, you will need to support scenarios in which different versions of a business process exist concurrently on a changing infrastructure. Managing this challenge

has implications throughout the project development lifecycle, not just for the runtime but also for the tools and methods used to define business processes within an enterprise.

You can manage the challenge of the dichotomy between business process cycles and product cycle by doing the following:

1) Reducing the impact of changes by modularization
2) Achieving middleware independence by defining the explicit process state
3) Monitoring and handling business exceptions

1) Reducing Impact by Modularization

Just as services can have different levels of granularity and permutations in the enterprise, processes also can have such granularity. This granularity appears when processes are designed as a composition of individual process modules. Each module offers a service interface and manages its own particular state internally. It then becomes much easier to change parts of the processes by developing new process modules that are selected from existing services using policies.

2) Achieving Middleware Independence with Explicit Process State

Current business process middleware engines maintain their process state internally. This dependency ties the process instances to the particular middleware engine, sometimes even to a particular version of the middleware. To avoid this, business process designers should elevate the explicit state beyond the engine level at each process step that leads to a waiting state until an external event arrives.

Thus, there is a need to be able to maintain and communicate state as distributed across the SOA. One particular programming model support for capturing these state descriptions is the set of specifications included in the WS-Resource Framework (as published on IBM developerWorks). These specifications allow the programmer to declare and implement the association between a Web service (a process module) and one or more identified, datatyped state components called WS-Resources.

3) Business Exceptions Monitoring and Handling

Even if the enterprise has spent a significant amount of time and effort to understand and model its business processes, undoubtedly unplanned business exceptions can still occur. A fully automated, services-oriented infrastructure that is capable of supporting any such exceptions to the business processes is unrealistic. This means that all business processes and their supporting infrastructure should be designed to allow manual recovery and control. Furthermore, for each business or technical domain, the organization should identify individuals that can handle such exceptions and act on the infrastructure. In most process and services identification modeling activities, the focus is on delivering mainstream models and a few variations. The business analysts must look at making the processes more granular so that unexpected variations and exceptions will be easier to handle in the operational environment.

6 Use Cases

This section outlines some business scenarios and how services may be defined to support them.

6.1 Appointment Scheduling

The scenario below is adapted from the September 2006 Scheduling V3 Ballot material under the Appointment Topic. Some sample service operation invocations are included in line in italics, some of which are discussed in a separate subsection below.

6.1.1 Physician Arranges For An Inpatient Stay

Orthopedic surgeon Dr. Sara Specialize determines, during an initial outpatient assessment, that patient Mr. Adam Everyman is a good candidate for a total hip replacement. Dr. Specialize arranges for Mr. Everyman to be put on the wait list for orthopedic surgery in the Good Health Hospital. The expected duration for the inpatient stay will be 6 days. The operation is to be performed by Dr. Specialize herself, approximately two months from now. In the meantime, all the necessary administrative requirements (e.g. insurance authorization) and medical tests (pre-operation screening) can be arranged by the GHH for Mr. Everyman.

After authorization and screening have been approved, the inpatient planner for the Good Health Hospital schedules a 6-day admission for Mr. Everyman *(Eligibility:checkEligibility, Scheduling:bookAppointment),* taking into account the availability of a hospital bed (*Resource:checkAvailability*, *Resource:reserveResource* or *Scheduling:getAvailableSlots, Scheduling:bookAppointment depending on scope of Scheduling*) in the surgical care unit in combination with the required session time in Dr. Specialize's OR schedule on the second day of the admission (*Scheduling:bookAppointment*). As a result of the scheduled admission, the hospital information system automatically triggers a notification of the new appointment (*Scheduling:notifyAppointment*)to the GHH Electronic Patient Record for Mr. Everyman.

The **Electronic Patient Record** provides a general view of all patient-related data to care providers within the Good Health Hospital, including specialists who use the system to prepare themselves for treating patients and to store their notes on a patient's medical history, evaluation and prognosis. Part of the patient record is an overview of all prior and planned admissions for a patient, as for outpatient encounters and other healthcare activities.

A request for his medical chart is sent to the central medical archive (*MedicalRecords:requestChart*) and/or (*RLUS:locateResource,retrieveResource*), which will deliver the chart on its regular delivery round 2 days prior to the scheduled visit.

The central medical archive runs an **Archive Management system**, that uses planned admissions to provide 'pick lists' for delivering patient charts from the archive (or from temporary locations where the chart resides) to the appropriate department on time. The data for the scheduled inpatient stay is used to provide the necessary information for

selecting the right chart (e.g. general inpatient chart) and knowing when and where it should be available.

The associated operation is placed in Dr. Specialize's OR session for day 2 of the stay.

There is a close relationship between inpatient logistics and the **OR Management system**, at least for patients with a scheduled operation during their stay. The way planning for the care units and the operating rooms are coordinated may differ among hospitals, but when the inpatient stay is scheduled this usually results in a status change (*Scheduling:notifyAppointment)* for the associated operation. This is placed in a specific OR session, for which ordering and planning may be refined later.

The hospital kitchen is informed of the dietary requirements for Mr. Everyman.

Food Management system is tightly linked to inpatient logistics, to make sure that sufficient meals are available for inpatients and that dietary requirements are met. Therefore the scheduled admission might be communicated to the food management system (*Scheduling:notifyAppointment*) in order to provide input for personnel planning and/or the ordering of ingredients. Note that the actual preparation of meals is usually bound to the eventual admission of the patient.

6.1.2 Patient Reschedules Outpatient Appointment

Mr. Adam Everyman has a conflict with the time that the new appointment was scheduled for the outpatient assessment of his right hip. Mr. Everyman calls the office of Dr. Specialize to reschedule the appointment for the outpatient assessment. Dr. Specialize's assistant reschedules the 10-minute slot to another 10-minute slot on the same day as the previously booked appointment (*Scheduling:rescheduleAppointment*). The appointment is rescheduled in Dr. Specialize's schedule for one of the outpatient clinics associated with the Good Health Hospital. As a result of the rescheduled appointment the hospital information system issues a notification of the reschedule to the GHH Electronic Patient Record for Mr. Everyman (*Scheduling:notifyAppointment*).

The request for his medical chart sent to the central medical archive, will be updated with the rescheduled time. (*MedicalRecords:updateChartRequest*),

The request for the recent orthopedic x-ray images sent to the radiology PACS, will be updated with the rescheduled time. (*Scheduling:notifyAppointment or Orders:updateOrder)*

The rescheduled appointment is sent to the patient tracking system to update its schedules. (*Scheduling:notifyAppointment*).

6.1.3 Patient Revises Outpatient Appointment

Mr. Everyman calls the office of Dr. Specialize to communicate the fact that he will be staying with his daughter until the scheduled appointment. If they need to contact him, they should call his daughter. Dr. Specialize's assistant revises the previously booked appointment with the new contact person's name and phone number. The appointment is revised in Dr. Specialize's schedule for the outpatient clinic where the appointment was booked. (*Scheduling:updateAppointment*) As a result of the revised appointment the

hospital information system issues a notification to update Mr. Everyman's daughter's phone number in the GHH Electronic Patient Record for Mr. Everyman. (*Person:updateDemographics*)

The revised appointment is sent to the patient tracking system to update its schedules. (*Scheduling:notifyAppointment*)

6.1.4 Physician Cancels Inpatient Stay for Patient

Mr. Everyman calls the office of Dr. Specialize to report that his pain is much less and he would like to postpone the surgery. Dr. Specialize examines Mr. Everyman (after a suitably scheduled out-patient appointment) and agrees that he does not need surgery at this time. Her assistant calls GHH to cancel the previously booked inpatient encounter appointment (*Scheduling:cancelAppointment*). As a result of the canceled appointment, the hospital information system of the Good Health Hospital notifies all interested parties of the current status. Each receiver then triggers internal processes to perform the following actions:

The entry of the scheduled appointment is canceled in Dr. Specialize's OR session schedule. (*Scheduling:cancelAppointment*)

The medical chart archives cancel the request for Mr. Everyman's medical chart. (*MedicalRecords:cancelChartRequest*)

The radiology PACS cancels the request of Mr. Everyman's recent orthopedic x-ray images. (*Scheduling:notifyAppointment or Orders:cancelOrder)*

The patient tracking system updates its schedules. (*Scheduling:notifyAppointment*)

6.1.5 Defining Services for the Appointment Scenario

Based on the four steps above, this section will define a sample service and operations based on the guidelines in section 4. This again is only intended to be illustrative of some of the concerns that may be considered. It is not intended to be the definitive solution.

6.1.5.1 Defining the Service

In this case, we start with the HL7 Domain “Scheduling”. This intuitively appears to be a good level for a single service, including all of the appointment related functions described in the scenario. It is a fairly cohesive and coherent set of business functionality that is not tightly coupled with other functional areas. One consideration is how abstract the service should be, i.e. specific to appointments or more general for any finite resources? The functionality of booking/reserving, rescheduling, canceling of slots etc. appears to be potentially common to many different resource types. In defining standard services, this approach would be recommended, where a generic scheduling service would be defined, with different semantic profiles for the different information content or resource types where necessary. For the sake of this example however, we will constrain the discussion to Appointments to keep within the scenario.

So, the Service Definition is:

- Name: Appointment Scheduling Service
- Description: This service provides capabilities to manage patient appointments. This includes the ability to check availability of appointments, book, reschedule and cancel appointments, as well as notifications to interested parties when an appointment is made or updated. It also provides capabilities to query for existing appointments and for open slots.

6.1.5.2 Defining the Interfaces

As indicated above, the service will provide capabilities for scheduling appointments, and also providing notifications. There are no real V3 artifacts to base this on, since the actual scheduling side has not yet been defined in V3. It is a subjective judgment whether to separate these capabilities into different interfaces, but since the basic requirements and nature of the interactions are different, they are separated in this definition. There may also be an "administrative" interface for starting and stopping the service, but we will not define that here, since we are concentrating on the business definition. So, two interfaces are defined:

Interface 1:

- Name: Appointment Scheduling
- Description: This interface provides capabilities to book, reschedule and cancel appointments.

Interface 2:

- Name: Scheduling Query
- Description: This interface provides capabilities to query for schedule information (free or scheduled slots etc.)

Interface 3:

- Name: Appointment Notification
- Description: This interface provides capabilities to request and receive notifications when an appointment is made or updated.

6.1.5.3 Defining the Operations

We will consider each of the above interfaces in turn, since the first two have only V2 precedent, and the latter V3.

Firstly, the Scheduling interface. V3 has not defined this area yet, so we look to V2 to direct us. Some of the Events identified provide good operations, i.e. Request Appointment (S01), Cancel (S04), Reschedule (S02) and so on. These are good, intuitive business actions with about the right granularity. Even though version V3 doesn't have corresponding interactions for these actions, the application role Appointment Requester also indicates a need for request appointment action. In defining the operations, some other considerations occur. Firstly, we could consider some of the process concerns, e.g.

what about validating that the patient is known and identified, and checking whether the patient is eligible for the service being requested. Although these are separate business actions which would be provided by other services, they could be included as part of a composite or process service within the overall Scheduling Service. There is no "correct" answer. The recommendation in cases like these is to define atomic operations for the scheduling without these additional capabilities and allow individual organizations and/or groups to define composite or process services over and above these. Providing the basic core services were standard, this would not be difficult to do.

So, we may define the following operations:

Interface 1: Name: Appointment Scheduling

- Scheduling:bookAppointment – reserves a specific appointment slot
- Scheduling:rescheduleAppointment – reschedules an appointment (i.e. cancels one and creates another new one)
- Scheduling:reviseAppointment – updates information associated with an Appointment without changing the slot itself.
- Scheduling:cancelAppointment – cancels an existing appointment

Other considerations for these operations could include the following:

1) A separate "confirmAppointment" operation could be defined, particularly where the service is being used interactively with a web UI front end.
2) At a logical definition level (SFM in HSSP terms), considering numbers of slots returned at a time and synchronous vs asynchronous processing would not be issues, so concerns such as halting a current request or continuation would not considered. For technical specifications, these need to be at least considered if an asynchronous solution is provided (a "deliberate" break of a synchronous connection would automatically stop an in process request, i.e. subject to appropriate reliable messaging solutions).

For the query interface, again we look to V2. Here we have the event: Schedule Query Message and Response (S25).

Interface 2: Name: Scheduling Query (covers both slot availability and appointment V3 topic queries)

- Query:requestAvailableSlots – requests for appointment slots meeting some supplied criteria
- Query:getExistingAppointments – requests for details of existing appointments for a patient, additional filtering criteria can be applied

For the Notification interfaces, we do have V3 precedent, so we can look to those artifacts. Ideally, an EDA approach (publish-and-subscribe mechanism) would be used for this functionality, which would enable topic-based subscription. Note that there are

emerging web service standards to explicitly support this paradigm. However, for now we will define a normal service interface.

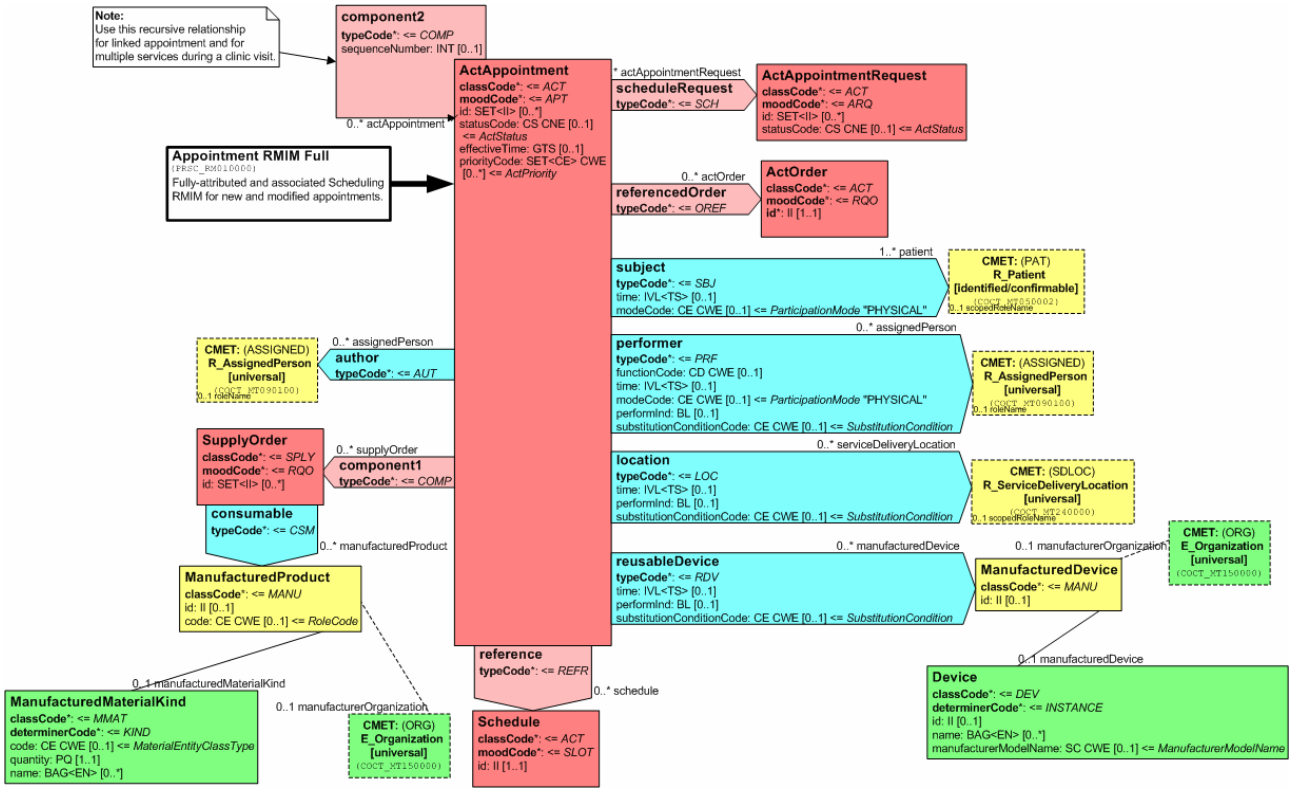
At the operation level, it probably makes sense to define a single operation for the notifications (which is a one-way operation initiated by the service) rather than separate ones for each event type. This is because the behavior and event metadata would be basically the same, as would much of the data content. Additionally, in an SOA world, the registration capability should be defined as an explicit operation.

- RegisterForAppointmentNotification
- NotifyAppointment

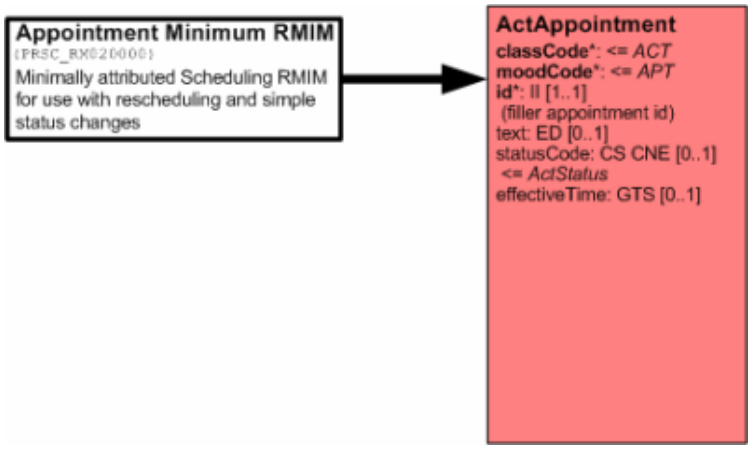
6.1.5.4 Identifying Message Content (Capability/Operation Input and Output)

The version 3 scheduling domain defines two data models which are used in deriving messages for the different notification messages. (new appt, revise appt, cancel and no show). The new and revise appointment notifications are based on the Appointment RMIM Full (PRSC_RM010000 and message type PRSC_MT010101UV01) which allows for quite detailed description of the appointments. Cancel and no show notifications are based on the Minimum Appointment RMIM (PRSC_RM020000). Both of the RMIMs are copied below from the September 2006 ballot.

Full appt RMIM:

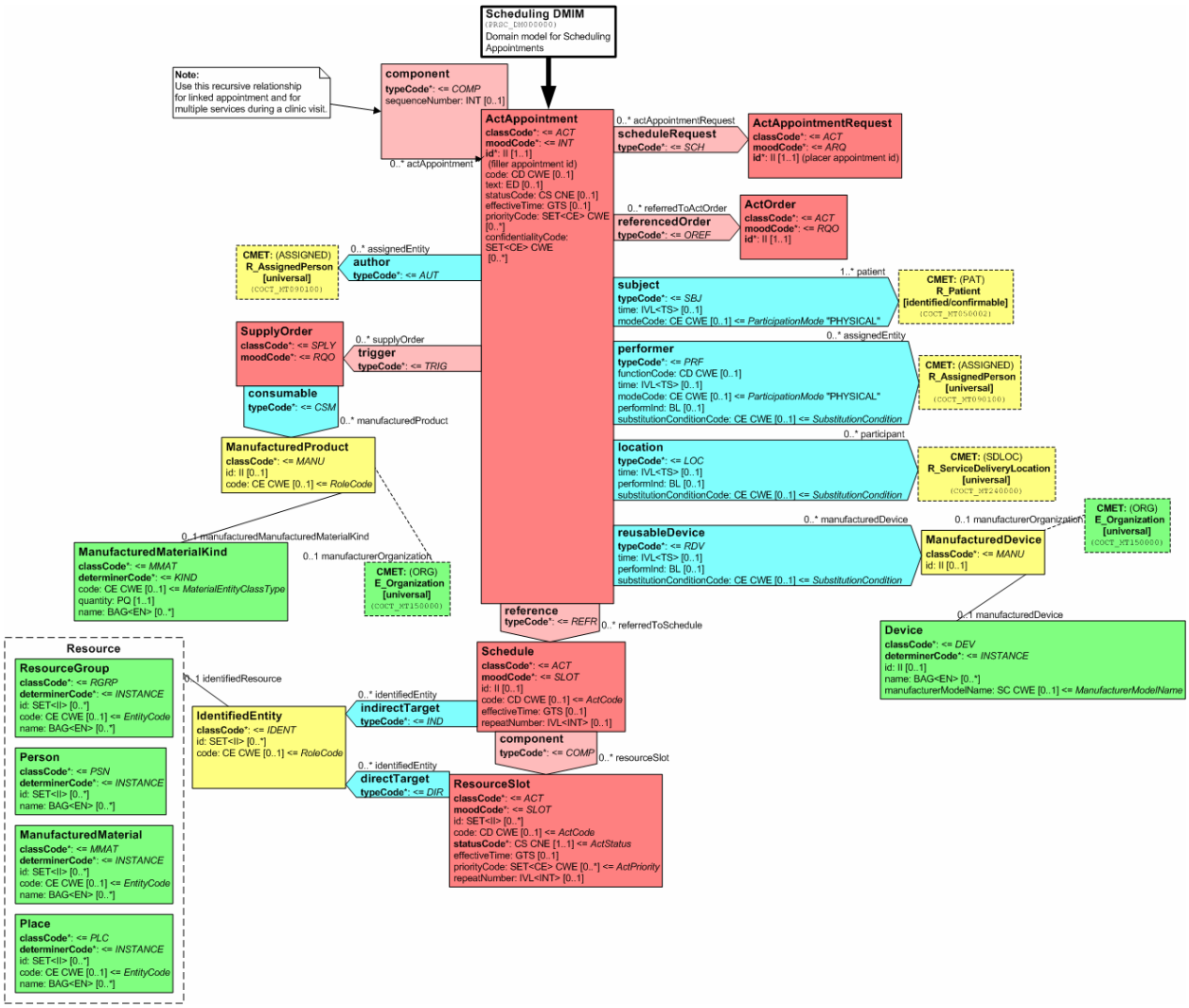


Minimum appt RMIM:



Following the methodology above, we start with the appropriate classes in scope from a DIM/D-MIM or most appropriate CIM/R-MIM level model available for this operation/capability if one is available. For example, if there was a RMIM for booking and rescheduling then this should have the most relevant information identified already.

The Appointment Minimum RMIM is a subset of the full appointment RMIM. If a single operation is defined for all notifications, then the data model used must be the full model to allow for all the needed data to be expressed. This would mean changing the only required participation (patient) to optional. Another approach would be to split the notification operation into two operations, one for dealing with new appointments and revisions (NotifyAppointment) and one for dealing with cancels and no-shows (NotifyCancelledAppointment)


Scheduling DMIM
(PRPC_DM000000)
Domain model for Scheduling Appointments
Note:
Use this recursive relationship for linked appointment and for multiple services during a clinic visit.
component
typeCode*: <= COMP
sequenceNumber: INT [0..1]
0..* actAppointment
ActAppointment
classCode*: <= ACT
moodCode*: <= INT
id*: II [1..1]
(filler appointment id)
code: CD CWE [0..1]
text: ED [0..1]
statusCode: CS CNE [0..1]
effectiveTime: GTS [0..1]
priorityCode: SET<CE> CWE [0..*]
confidentialityCode: SET<CE> CWE [0..*]
0..* actAppointmentRequest
scheduleRequest
typeCode*: <= SCH
ActAppointmentRequest
classCode*: <= ACT
moodCode*: <= ARQ
id*: II [1..1] (placer appointment id)
0..* referredToActOrder
referencedOrder
typeCode*: <= OREF
ActOrder
classCode*: <= ACT
moodCode*: <= RQO
id*: II [1..1]
CMET: (ASSIGNED) R_AssignedPerson [universal]
(COCT_MT090100)
0..* assignedEntity
author
typeCode*: <= AUT
1..* patient
subject
typeCode*: <= SBJ
time: IVL<TS> [0..1]
modeCode: CE CWE [0..1] <= ParticipationMode "PHYSICAL"
CMET: (PAT) R_Patient [identified/confirmable]
(COCT_MT050002)
SupplyOrder
classCode*: <= SPLY
moodCode*: <= RQO
0..* supplyOrder
trigger
typeCode*: <= TRIG
0..* assignedEntity
performer
typeCode*: <= PRF
functionCode: CD CWE [0..1]
time: IVL<TS> [0..1]
modeCode: CE CWE [0..1] <= ParticipationMode "PHYSICAL"
performInd: BL [0..1]
substitutionConditionCode: CE CWE [0..1] <= SubstitutionCondition
CMET: (ASSIGNED) R_AssignedPerson [universal]
(COCT_MT090100)
consumable
typeCode*: <= CSM
0..* manufacturedProduct
0..* participant
location
typeCode*: <= LOC
time: IVL<TS> [0..1]
performInd: BL [0..1]
substitutionConditionCode: CE CWE [0..1] <= SubstitutionCondition
CMET: (SDLOC) R_ServiceDeliveryLocation [universal]
(COCT_MT240050)
ManufacturedProduct
classCode*: <= MANU
id: II [0..1]
code: CE CWE [0..1] <= RoleCode
0..1 manufacturedManufacturedMaterialKind
ManufacturedMaterialKind
classCode*: <= MMAT
determinerCode*: <= KIND
code: CE CWE [0..1] <= MaterialEntityClassType
quantity: PQ [1..1]
name: BAG<EN> [0..*]
0..1 manufacturerOrganization
CMET: (ORG) E_Organization [universal]
(COCT_MT150000)
0..* manufacturedDevice
reusableDevice
typeCode*: <= RDV
time: IVL<TS> [0..1]
performInd: BL [0..1]
substitutionConditionCode: CE CWE [0..1] <= SubstitutionCondition
0..1 manufacturerOrganization
ManufacturedDevice
classCode*: <= MANU
id: II [0..1]
CMET: (ORG) E_Organization [universal]
(COCT_MT150000)
0..1 manufacturedDevice
Device
classCode*: <= DEV
determinerCode*: <= INSTANCE
id: II [0..1]
name: BAG<EN> [0..*]
manufacturerModelName: SC CWE [0..1] <= ManufacturerModelName
reference
typeCode*: <= REFR
0..* referredToSchedule
Schedule
classCode*: <= ACT
moodCode*: <= SLOT
id: II [0..1]
code: CD CWE [0..1] <= ActCode
effectiveTime: GTS [0..1]
repeatNumber: IVL<INT> [0..1]
component
typeCode*: <= COMP
0..* resourceSlot
Resource
ResourceGroup
classCode*: <= RGRP
determinerCode*: <= INSTANCE
id: SET<II> [0..*]
code: CE CWE [0..1] <= EntityCode
name: BAG<EN> [0..*]
Person
classCode*: <= PSN
determinerCode*: <= INSTANCE
id: SET<II> [0..*]
name: BAG<EN> [0..*]
ManufacturedMaterial
classCode*: <= MMAT
determinerCode*: <= INSTANCE
id: SET<II> [0..*]
code: CE CWE [0..1] <= EntityCode
name: BAG<EN> [0..*]
Place
classCode*: <= PLC
determinerCode*: <= INSTANCE
id: SET<II> [0..*]
code: CE CWE [0..1] <= EntityCode
name: BAG<EN> [0..*]
0..1 identifiedResource
IdentifiedEntity
classCode*: <= IDENT
id: SET<II> [0..*]
code: CE CWE [0..1] <= RoleCode
0..* identifiedEntity
indirectTarget
typeCode*: <= IND
0..* identifiedEntity
directTarget
typeCode*: <= DIR
ResourceSlot
classCode*: <= ACT
moodCode*: <= SLOT
id: SET<II> [0..*]
code: CD CWE [0..1] <= ActCode
statusCode*: CS CNE [1..1] <= ActStatus
effectiveTime: GTS [0..1]
priorityCode: SET<CE> CWE [0..*] <= ActPriority
repeatNumber: IVL<INT> [0..1]

Having one general operation gives many advantages. One disadvantage is for the implementer who needs to go into the message payload / service call payload to figure out what should be done. For example even thought the information content for no show and cancel is the same, the functionality that the developer needs to implement is quite different (no-show might have billing implications, cancel might initiate some process to utilize the freed resource)? One reason for going with a single data model could be to get a single schema on the XML level. This however makes the interface implementation of the involved applications more complicated because of the large number of optional structures that the applications will have to deal with. Below are two options for the notification specification, one with a single data model and one with 2 data models.

Approach 1: Start from the full appointment RMIM and start looking for relevant classes and CMETs.

NotifyAppointment operation input:

Problem: how do we identify which type of notification the NotifyAppoiment call is notifying about? There is no data element for this information since in the V3 approach the interaction expresses what type of notification is in question (new, revise, cancel, noshow).

Relevent classes and CMETs are:

ActAppointment class, Patient CMET, Service delivery location CMET, Schedule and resourceSlot classes and resource entities participating in those classes. ResourceSlot effectiveTime or slotID indicates the time be scheduled.

Output: none

Approach 2: (alternative, use existing schemas as much as possible)

Split notifications into two types:

NotifyAppointmentOrChange

NotifyAppointmentCancelOrNoshow

Input for NotifyAppointmentOrChange: If the notification function covers these, take RMIM Full Appointment (PRSC_RM010000) and create XML schema from this diagram manually.

For cancel and no show similarly take the minimum RMIM and manually create XML schema based on the HMD.

Scheduling:bookAppointment (based on SerAPI work from Finland)

Relevant classes and CMETs for defining input:

Patient CMET, Service delivery location CMET, Schedule and resourceSlot classes and resource entities participating in those classes. ResourceSlot effectiveTime or slotID indicates the time to be scheduled.

Output: Schedule, Slot, resource entities participating in those classes and the patient

(loosely coupled output, in a tightly coupled solution a boolean output would be sufficient)

6.1.5.4 Identifying Exceptions

Consider receiver responsibilities (interactions) for exceptions

6.1.5.5 WSDL Specifications (only bookAppointment is expanded)

HEADER

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="urn:hl7soa:Scheduling"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:hl7soa="urn:hl7soa:Scheduling"
        xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        name="Schedulingv1">
```

TYPES

```
<types>
<schema elementFormDefault="qualified" targetNamespace="urn:hl7-org:v3"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:hl7soa="urn:hl7soa:Scheduling">
<include schemaLocation="COCT_MT150000UV04.xsd" />
<include schemaLocation="datatypes-base.xsd" />
<element name="AppointmentRequest">
<complexType>
<sequence>
<!-- basic data for Scheduling request: patient, scheduleId (from which schedule
time is booked - note: not the only option to identify schedule - resources etc.), time
to be booked (slot id or interval of time)  -->
<xs:element name="subject" type="COCT_MT050002UV04.Patient" nillable="true"
maxOccurs="unbounded"/>
<xs:element name="scheduleId" type="II" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="resourceSlotId" type="II" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="effectiveTime" type="IVL_TS" minOccurs="0"
maxOccurs="unbounded" />
```

<!-- additional data for Scheduling request: resources (in addition to actual schedule id, place, resourceGroup, person, manufacturedmaterial -->

<xs:element name="placeId" type="II" minOccurs="0" maxOccurs="unbounded" />

<xs:element name="resourceGroupId" type="II" minOccurs="0" maxOccurs="unbounded" />

<xs:element name="resourcePerson" type="IdentifiedPerson" minOccurs="0" maxOccurs="unbounded" />

<xs:element name="manufacturedMaterialId" type="II" minOccurs="0" maxOccurs="unbounded" />

<element name="noteText" type="string" minOccurs="0"/>

<!-- administrative data for Scheduling request omitted from example -->

</sequence>

</complexType>

<!-- simple person identifier and name -->

</element>

<xs:complexType name="IdentifiedPerson">

<xs:sequence>

<xs:element name="id" type="II" minOccurs="0" maxOccurs="unbounded" />

<xs:element name="name" type="EN" minOccurs="0" maxOccurs="unbounded" />

</xs:sequence>

</complexType>

<!-- etc. (types for AppointmentResponse and others) -->

</schema>

</types>

MESSAGES

<message name="BookAppointmentRequestMessage">

<part name="parameters" element="hl7soa:**AppointmentRequest**"/>

</message>

<message name=" BookAppointmentResponseMessage">

<part name="parameters" element="hl7soa:**AppointmentResponse**"/>

</message>

<message name="ExceptionMessage">

<part name="fault" element="hl7soa:Exception"/>

```
</message>
ETC.
```

PORTTYPE

```
<portType name="AppointmentScheduling">
<operation name="BookAppointment">
<input message="BookAppointmentRequestMessage"/>
<output message="BookAppointmentResponseMessage"/>
<fault name="Exception" message=""/>
</operation>
<operation name="RescheduleAppointment">
<input message=""/>
<output message=""/>
<fault name="Exception" message="hl7soa:ExceptionMessage"/>
</operation>
<operation name="ReviseAppointment">
<input message=""/>
<output message=""/>
<fault name="Exception" message="hl7soa:ExceptionMessage"/>
</operation>
<operation name="CancelAppointment">
<input message=""/>
<output message=""/>
<fault name="Exception" message="hl7soa:ExceptionMessage"/>
</operation>
</portType>
<portType name="SchedulingQuery">
<operation name="RequestAvailableSlots">
<input message=""/>
<output message=""/>
<fault name="Exception" message="hl7soa:ExceptionMessage"/>
</operation>
<operation name="GetExistingAppointments">
```

<input message=""/>

<output message=""/>

<fault name="Exception" message="hl7soa:ExceptionMessage"/>

</operation>

</portType>

BINDING+SERVICE

<binding name="APRServiceSOAPBinding" type="hl7soa:APRServiceOperations">

<wsdlsoap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>

<operation name="Get_pAPR">

<wsdlsoap:operation soapAction="urn:hl7soa:APR#Get_pAPR"/>

<input>

<wsdlsoap:body use="literal"/>

</input>

<output>

<wsdlsoap:body use="literal"/>

</output>

<fault name="Exception">

<wsdlsoap:fault name="Exception" use="literal"/>

</fault>

</operation>

</binding>

<service name="APRService">

<port name="APRServiceSOAPPort" binding="hl7soa:APRServiceSOAPBinding">

<wsdlsoap:address location="http://localhost/APRService/services/APRService"/>

</port>

</service>

6.2 Billing Example (Another Approach)

This section presents another approach that is being taken in the Finance domain for being cognizant of the above service design principles in current V3 work, specifically around billing.[4]

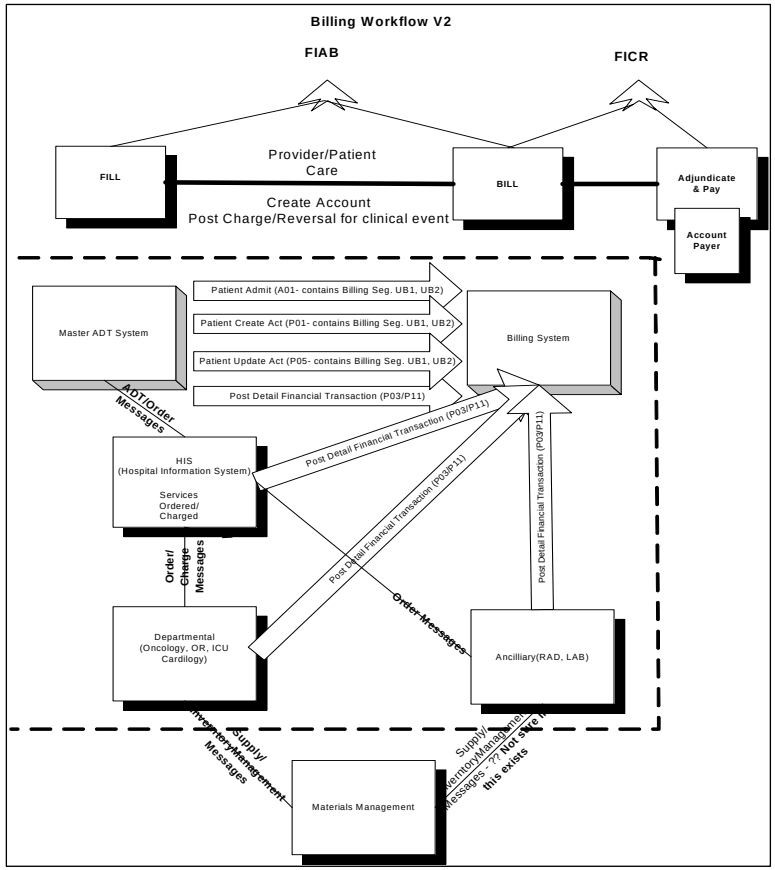


Figure 3: Legacy System

[4] Although not critical to understanding, one or two of the diagrams in this section have been compressed and cannot be easily interpreted. Zooming in to 150% r 200% should be sufficient, but original versions are also be available on the HSSP Wiki at http://hssp-implementation.wikispaces.com/ActiveWorkRoot.

Version 2 financial management domain messages are typically coarse-grained and tightly coupled. That is, the messages contained all the information needed for performing one or more business processes, e.g., the A01 sent by the ADT system notifies all systems participating in the scheduled encounter about all patient and encounter information that any of them might need.

It is up to each system to capture the information relevant to its business purposes. Detailed information that is not managed by any one of these systems is assumed to be captured by the system charged with managing it, and only minimal linking data would be captured by ancillary systems.

BAR^P01^BAR P01	Add Billing Account
MSH	Message Header
[{ SFT }]	Software Segment
[UAC]	User Authentication Credential
EVN	Event Type
PID	Patient Identification
[PD1]	Additional Demographics
[{ ROL }]	Role
{	--- VISIT begin
[PV1]	Patient Visit
[PV2]	Patient Visit - Additional Info
[{ ROL }]	Role
[{ DB1 }]	Disability Information
[{ OBX }]	Observation/Result
[{ AL1 }]	Allergy Information
[{ DG1 }]	Diagnosis
[DRG]	Diagnosis Related Group
[{	--- PROCEDURE begin
PR1	Procedures
[{ ROL }]	Role
}]	--- PROCEDURE end
[{ GT1 }]	Guarantor
[{ NK1 }]	Next of Kin/Associated Parties
[{	--- INSURANCE begin
IN1	Insurance
[IN2]	Insurance - Additional Info.
[{ IN3 }]	Insurance - Add'l Info. - Cert.
[{ ROL }]	Role
}]	--- INSURANCE end
[ACC]	Accident Information
[UB1]	Universal Bill Information
[UB2]	Universal Bill 92 Information
}	--- VISIT end

Figure 4: Financial Patterns

Prior to consideration of design for SOA, Version 3 financial communications merely replicated the same patterns.

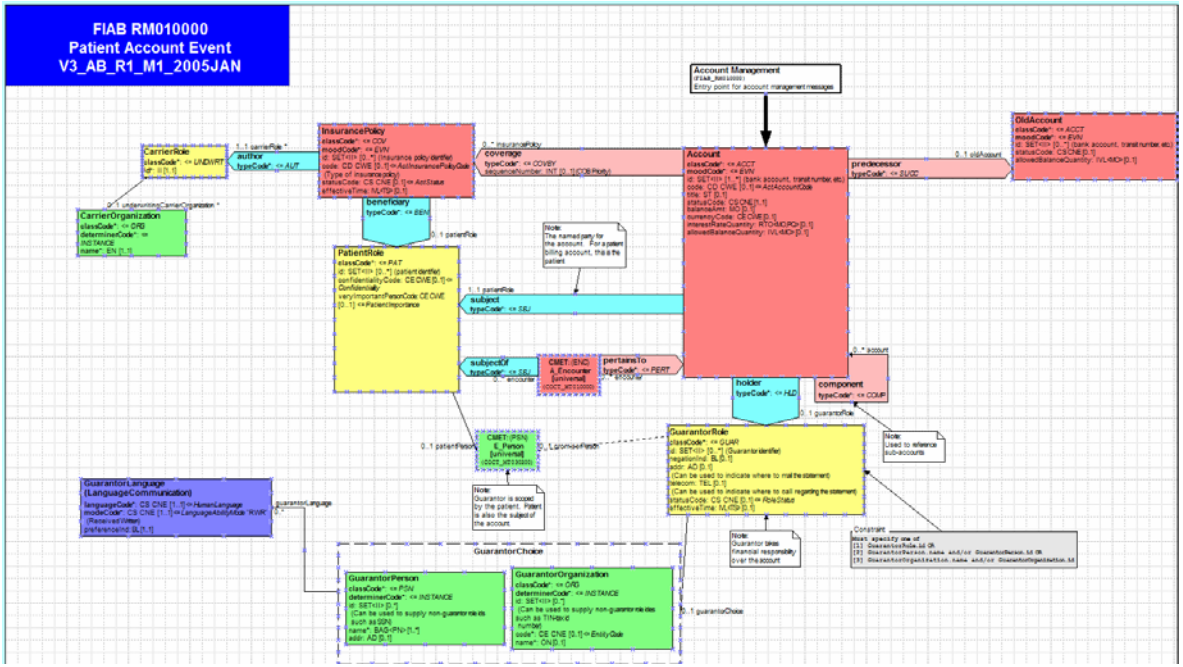


Figure 5: V3R2 Financial management domain model

Version 3 Release 2 of the financial management domain is intended to modularize discrete business requirements to permit the composition of messages that support both coarse and fine grained communication patterns whether these are used within tightly or loosely coupled environments.

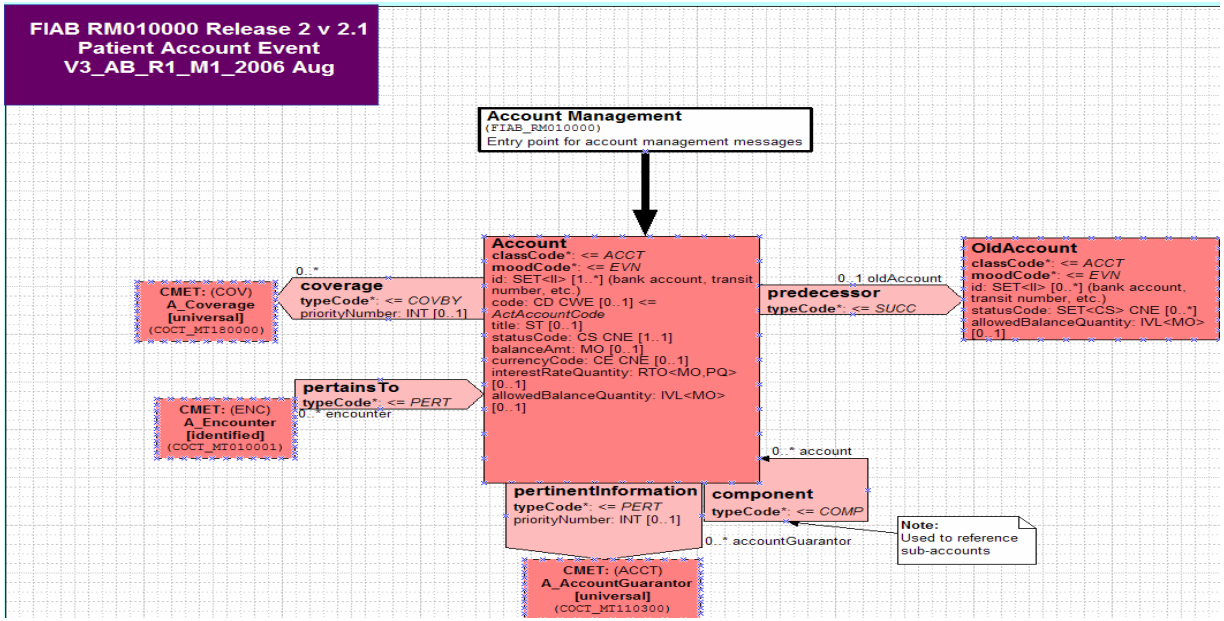


Figure 6: V3R3 Patient account events model

Analysis of the business processes pertaining to the life cycle of a patient billing account provided insight as to the junctures at which critical information about this process would be captured, and the virtual location of this “data of record” in a hypothetical environment in which each business process operated with total autonomy. This provided the boundaries for the modules.

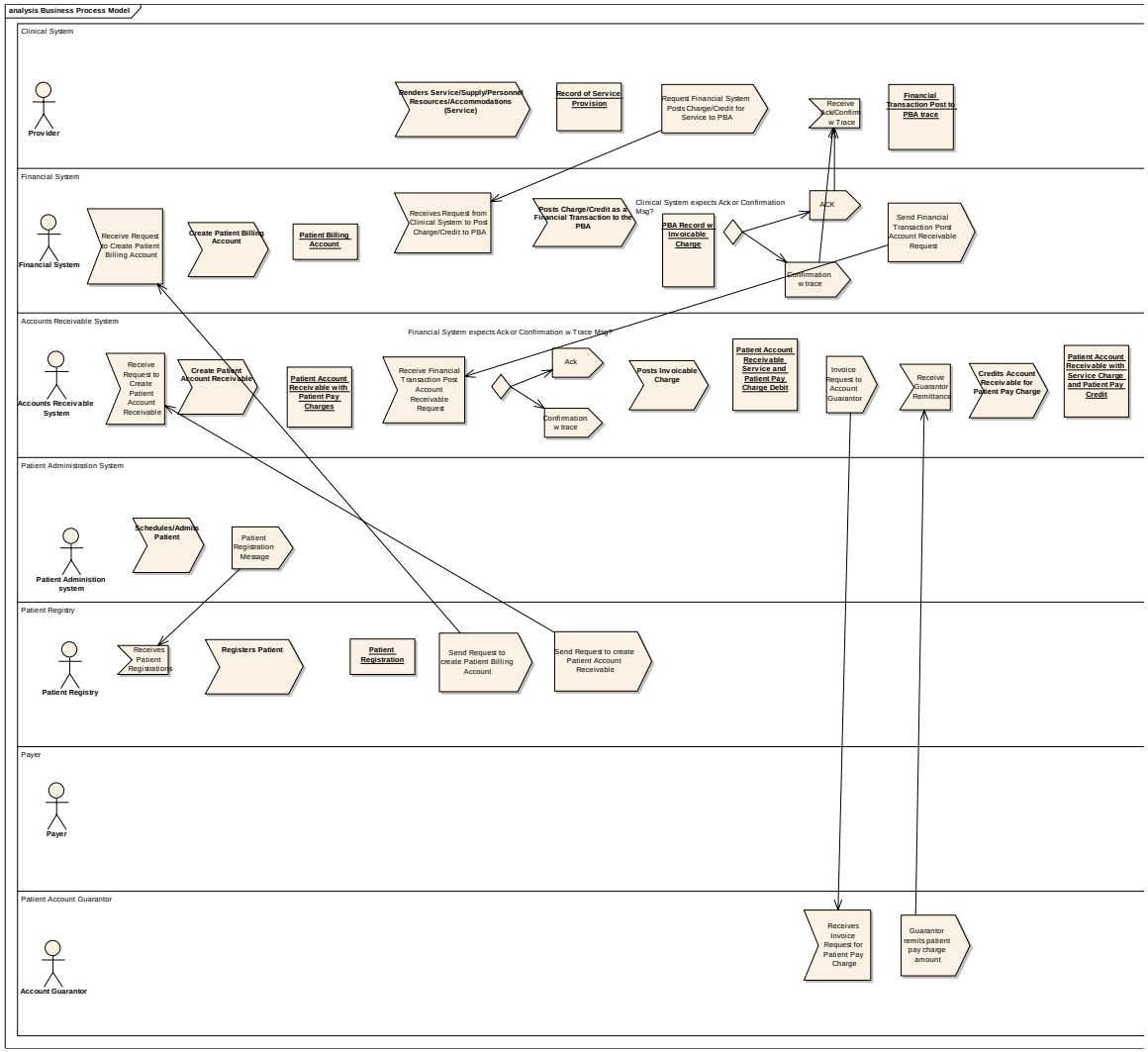


Figure 7: Business Process Model

For example, patient demographics are captured at admission, and may be augmented with information derived from covered party information captured during eligibility verification. The patient demographics might be stored in (1) a patient registry in a loosely coupled federated system supported by a very cohesive, complete, and fine grained service; (2) a master file in a tightly coupled legacy system supported by coarse grained, more or less complete services; or (3) within the patient's electronic or non-electronic

health record in system supported by a "mixed-bag" of services. To arrive at the appropriate level of *design time* granularity, it is best to develop fine grained services (for modularity) that support loosely coupled systems (for inclusive support of all data requirements), which of course, would not suit implementation (too much overhead, etc, see above). For implementation purposes, you can then compose and constrain the modules so that environments requiring either fine or coarse grained services may be supported.

For example, encounter information may be captured at scheduling or be contained within a referral message. The coverage information may be captured at admission from the patient, or obtained via an eligibility verification transaction. The guarantor information may be collected at admission or derived from an eligibility verification response from a payer. In all these cases, there seem to be modules of information captured at typical steps in the business process. However, the location in which the "data of record" is stored may be quite diverse. The information may be transferred to the financial system within an admission or scheduling notification (coarse grained service environment), or obtained by the financial system from the patient registry based on keys sent in the notifications (fine grained service environment). Design the modules as if each were managed and accessed independently. Compose and constrain modules to support business process configuration needs.

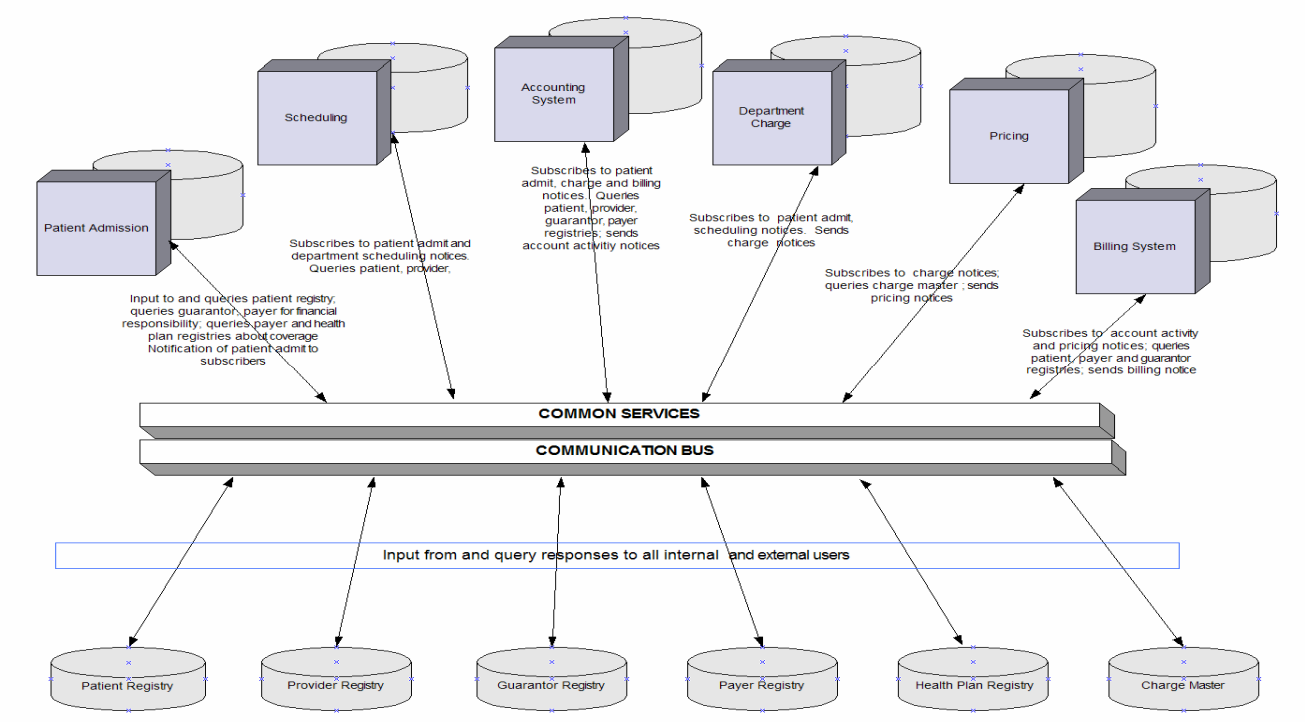


Figure 8: Common Services and the communication bus

Based on this analysis, Release 2 Version 3 messages within the Financial Management Domain are being designed to permit coupling at the appropriate level of granularity given the implementers' enterprise configuration. To the extent possible, each module is constructed as a CMET. Each CMET has a number of variants that support a range of use cases: From a tightly coupled environment which requires only minimal "key" information such as the identifiers needed to locate an entity in a registry (*"Skinny" CMETs such as the identified, indentified-confirmable, informational, identified informational, minimal, and contactable variants)* to the most robust transfer of information needed in a loosely coupled environment where systems do not have access to repositories for that information (*"Fat" CMETs such as the basic, enhanced, business-specific, e., and universal variants)* .

In addition, where there are similar structures to the modules, these are abstracted to permit reuse of similar semantic constructs. For example, a patient billing account is a constraint on an account with a "holder" played by the role of the owner and responsible party scoped by the entity recognizing the owner and played by an owning entity. The same structure can be reused for a guarantor, a payer, a payee, and a cost account.

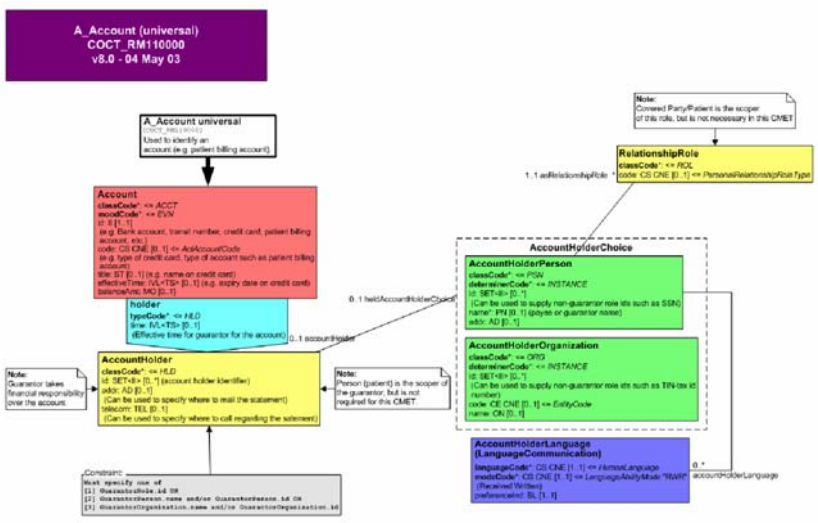


Figure 9: Universal account model

At the most generic level, there can be a CMET that includes all “flavors” of like CMETs to permit selection of the appropriate one for the business circumstances. A currently balloted example is the A_Billable universal, which supports selection of all types of billable services, and could easily be extended to include new ones.

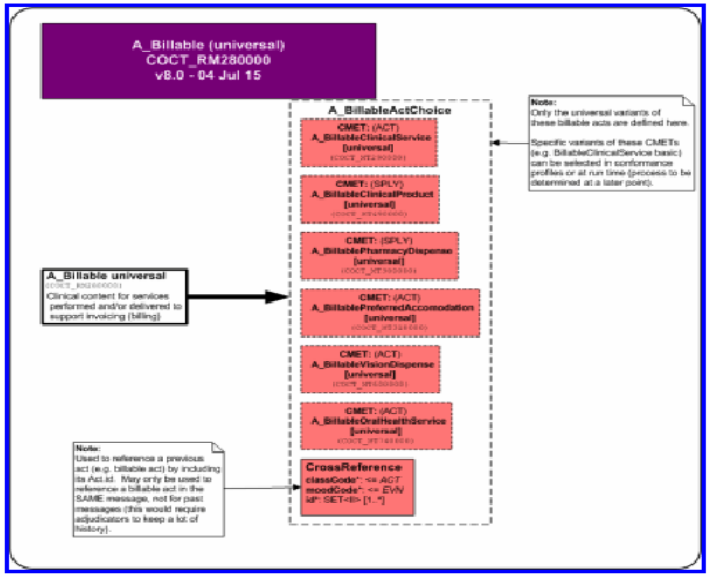
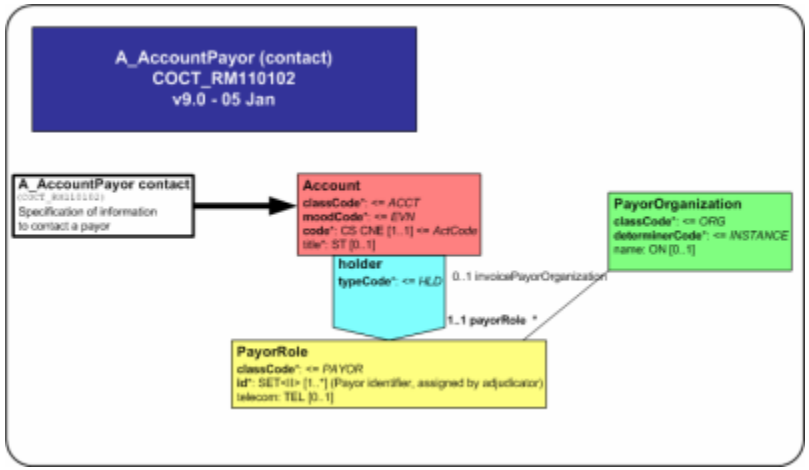
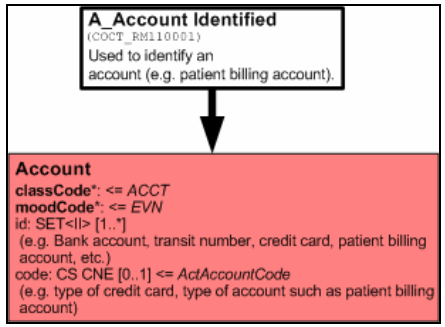
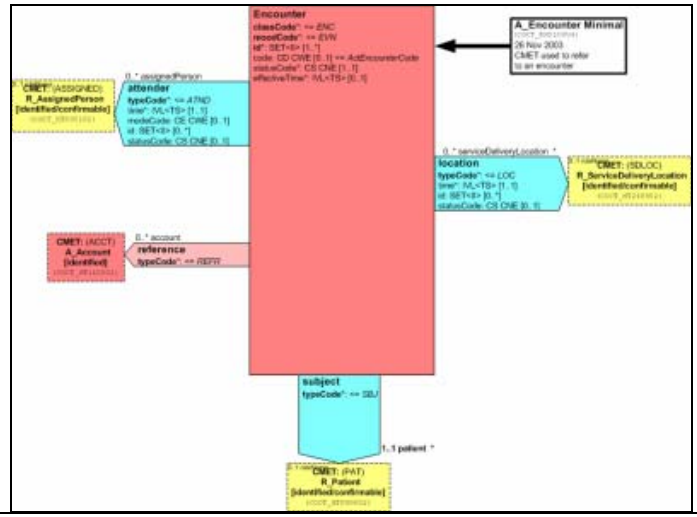
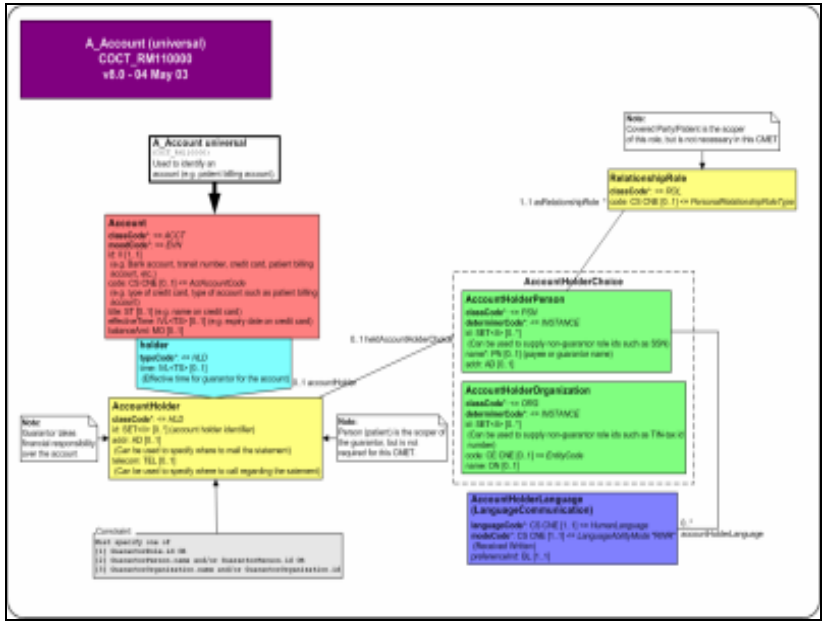
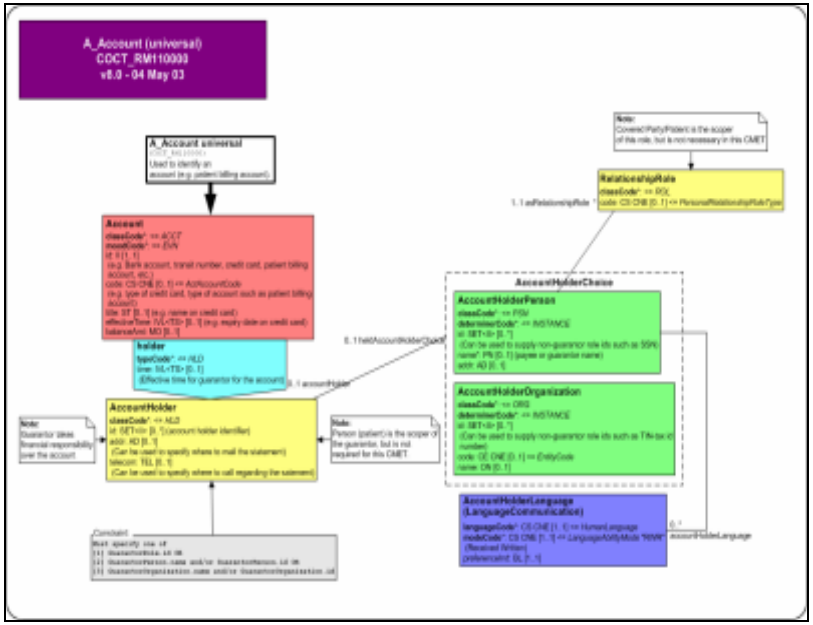


Figure 10: Universal billable model

This design approach permits the construction of communication interfaces that can be constrained at run time to the appropriate level of granularity required by the degree of service interdependencies (coupling).

	Fine Grained	Coarse Grained
Tightly Coupled	Use constrained "typed" CMET RMIMs as the interface, e.g., A_AccountPayor (contact) 	Skinny CMETs associated with the focal business object, optionally associated as needed at run time  
Loosely Coupled	Use loose CMET RMIMs that can be constrained for type and robustness of data content at run time depending on coupling requirements.	Fat CMETs associated with the focal business object, which can be dropped or constrained at run time depending on coupling requirements. Using a choice box with types of A_Account universals, e.g., A_AccountGuarantor, A_Account


A_Account (universal)
COCT_RM110000
v8.0 - 04 May 03
A_Account universal
Used to identify an account (e.g. patient billing account).
Account
holder
AccountHolder
AccountHolderChoice
AccountHolderPerson
AccountHolderOrganization
AccountHolderLanguage (LanguageCommunication)
RelationshipRole


A_Account (universal)
COCT_RM110000
v8.0 - 04 May 03
A_Account universal
Used to identify an account (e.g. patient billing account).
Account
holder
AccountHolder
AccountHolderChoice
AccountHolderPerson
AccountHolderOrganization
AccountHolderLanguage (LanguageCommunication)
RelationshipRole

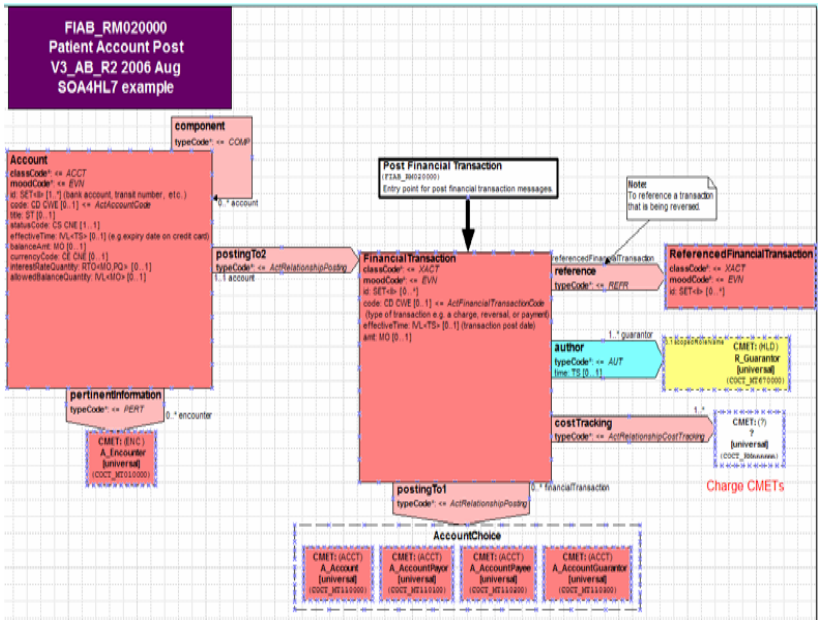
		

Table 15: Coupling and Granularity model for billing use case

7 Profiling and Conformance

The concept of profiling of Services has been defined within HSSP to provide a means to provide a fairly generic service description with more specific implementations. Functional profiles are defined to provide a means of sub-setting functional capabilities, semantic profiles as a means of supporting different information models within the same operations, and conformance profiles which provide a combination of the two. Further details can be found in the overall SSF.

These should also be considered for Services defined using this methodology. Referring to the Appointments scenario in section 6 above, it would be possible to define a generic "Scheduling" service, and define functional and semantic profiles to deal specifically with Outpatient Appointments.

8 Appendix A – Relationship to HSSP SSF

This section depicts the relationship between the SOA4HL7 methodology and the overall HSSP Service Specification Framework. The first diagram shows SOA4HL7 within the overall SSF context. The other two diagrams show drill downs of the sub-processes for creating the SFM and RFP that are part of the main SSF. The drilldown for the SOA4HL7 is included above in section 3.

8.1 Overall SSDF Process (including SOA4HL7)

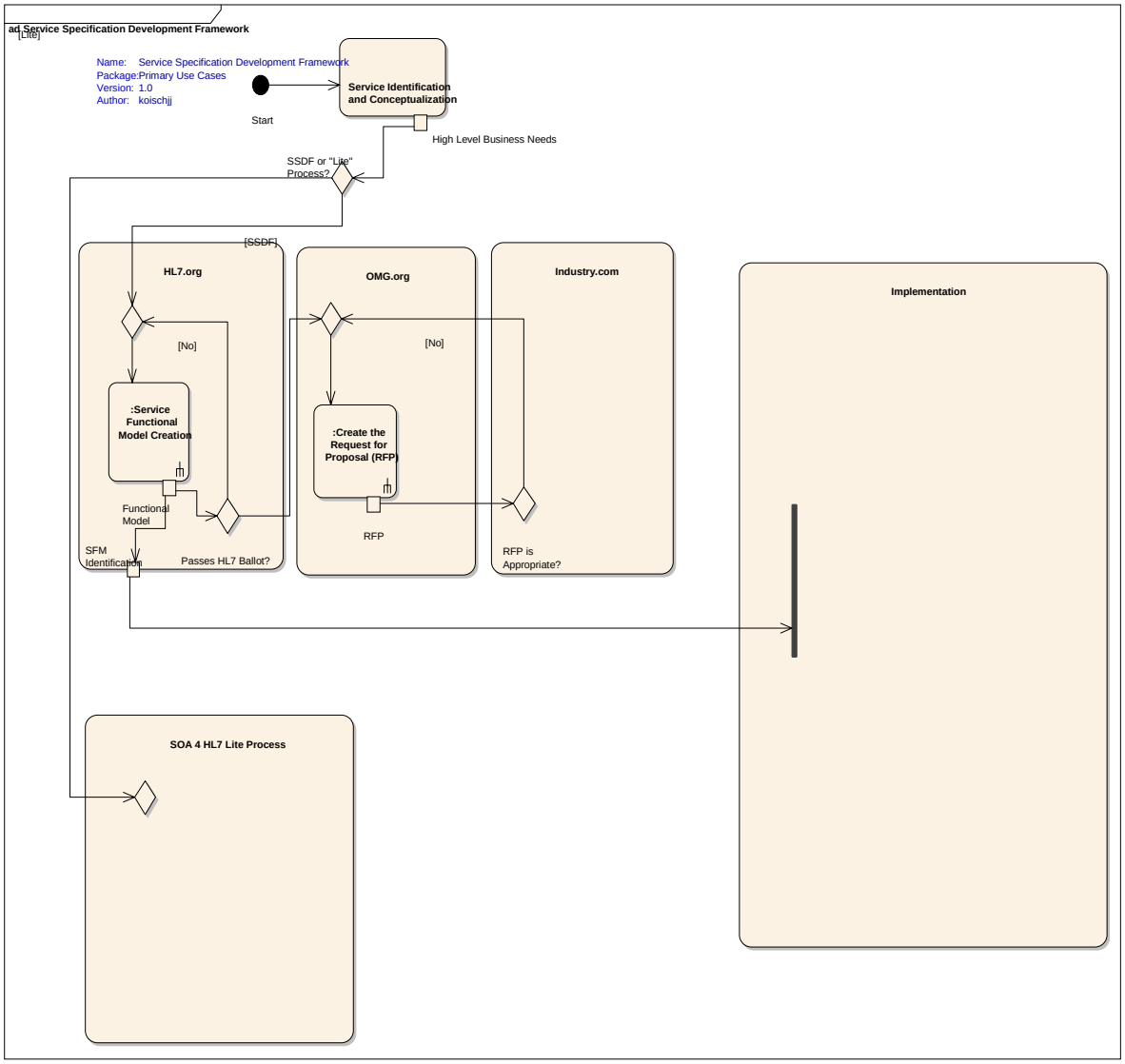


Figure 11: Overall SSF Process (including SOA4HL7)

8.2 Produce SFM (part of main SSDF)

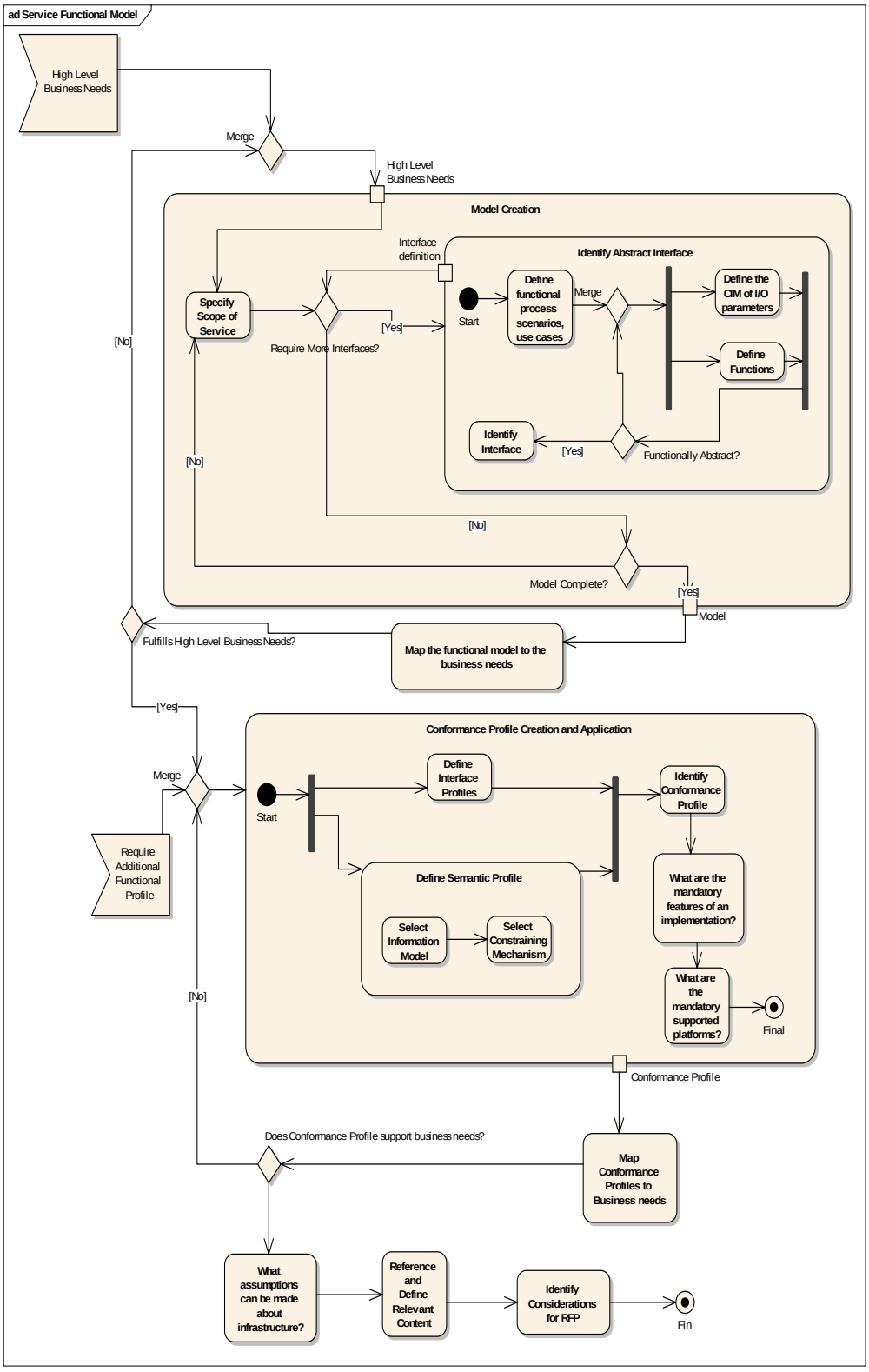


Figure 12: Produce SFM (part of main SSF)

8.3 Produce RFP (part of main SSF)

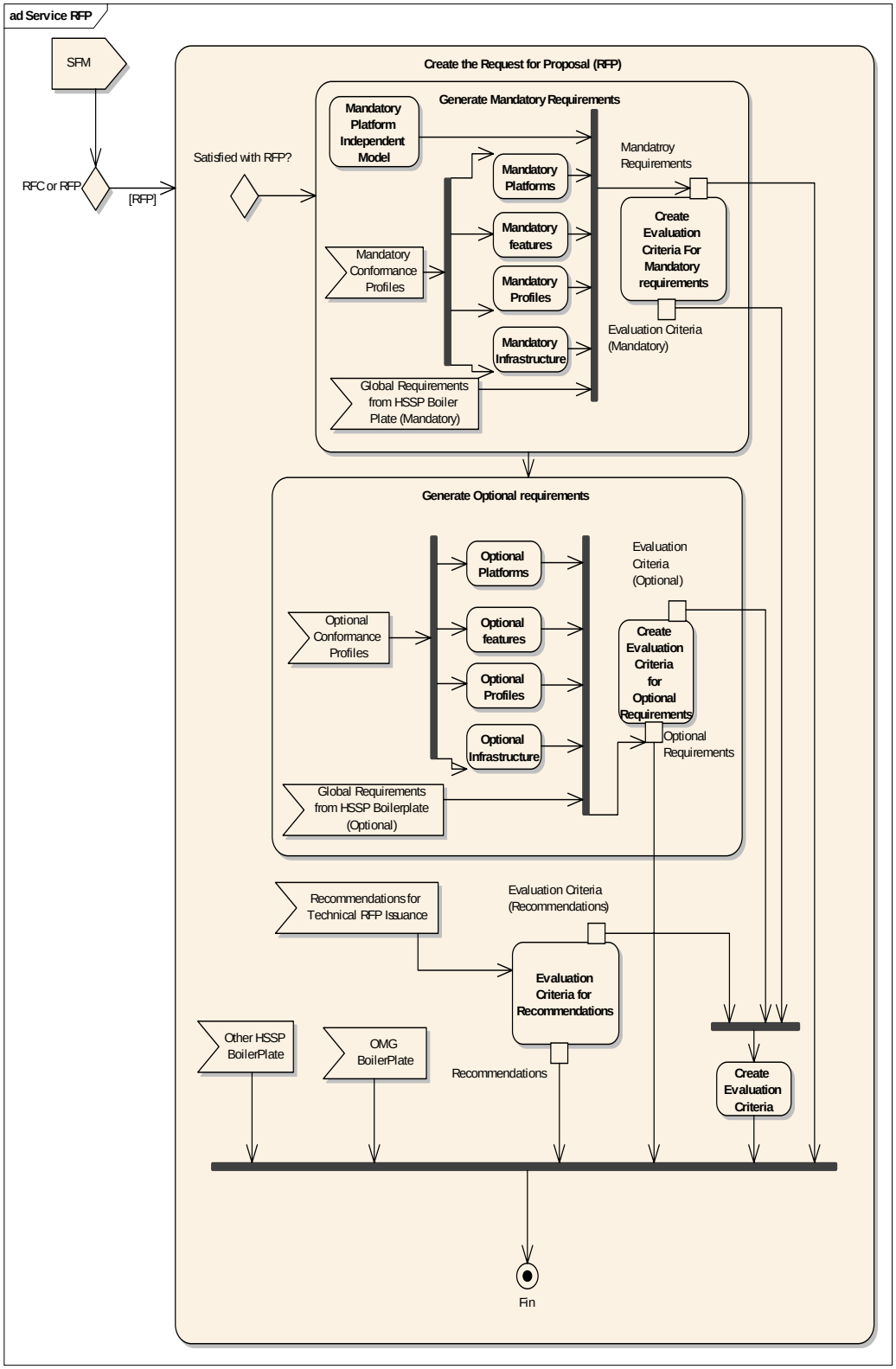


Figure 13: Produce RFP (part of main SSF)

9 Appendix B - References

CBDI Forum (SOA Practice) - http://www.cbdiforum.com/index.php3

Berners-Lee T. Axioms of design.

http://www.w3.org/DesignIssues/Principles.html#KISS

Papazoglou M, van den Heuvel W-J. Service-Oriented Design and Development Methodology. Int.J. of Web Engineering and Technology (IJWET), 2006. http://infolab.uvt.nl/pub/papazogloump-2006-88.pdf

Perepletchikov M, Ryan C, Tari Z. The Impact of Software Development Strategies on Project and Structural Software Attributes in SOA. Second INTEROP Network of Excellence Dissemination Workshop (INTEROP'05). 2005. Ayia Napa, Cyprus.

Stojanovic Z. A Method for Component-Based and Service-Oriented Software Systems Engineering. Delft University of Technology, 2005.

Zimmermann O, Schlimm N, Waller G, Pestel M. Analysis and Design Techniques for Service-Oriented Development and Integration. http://www.perspectivesonwebservices.de/download/INF05-ServiceModelingv11.pdf