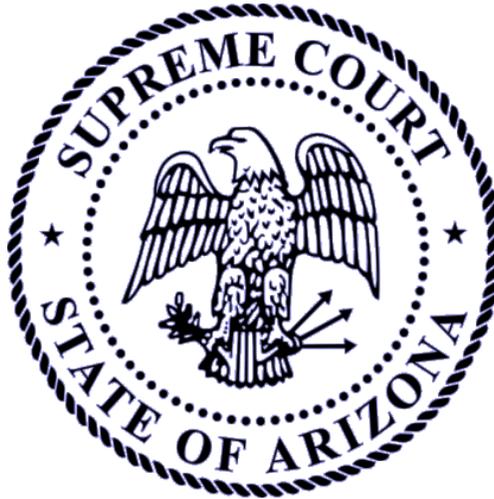


# Administrative Case Event System (ACES) Project Strategic Design Document



Authored by	Steele Price
Version	1.0
Dated	6/1/2016
File Name	ACES Strategic Design.docx

## Review Control and Change Control

### Review Control

Version	Date	Reviewer
1.0	June 21, 2016	Brian Heady

### Change Control

Version	Date	Change	Actioned By
1.0	June 1, 2016	Initial Document	Steele Price
1.1	June 21, 2016	Applied several Edits, Added Examples for Clarity	Steele Price
1.2	June 29, 2016	Reworded some phrases, added examples	Steele Price
1.3	July 1, 2016	Modified for the newest Aggregate Changes	Steele Price

## 1 CONTENTS

1	Contents .....	3
2	Overview .....	5
2.1	Introduction .....	5
2.2	Assumptions .....	5
3	Process Design .....	6
3.1	Process Flow .....	6
3.2	Major Design Elements .....	7
3.2.1	Design Goals .....	7
3.2.2	Public Interface .....	7
3.2.3	Domain Adapter .....	8
3.2.4	Projected View .....	8
3.2.5	Domain Aggregate .....	8
3.2.6	Domain Entity .....	9
3.2.7	Value Object .....	9
3.2.8	Domain Services .....	9
3.2.9	Domain Repository .....	9
3.2.10	Event Store .....	10
3.2.11	Low Level Repository .....	11
3.2.12	External Elements .....	12
4	Message Processing .....	13
4.1	Command Message .....	13
4.2	Event Message .....	13
4.3	Document Message .....	13
4.4	Query Message .....	13
4.5	Communication Matrix and Message Format .....	14
5	Design Pattern Reference .....	15
6	Works Cited .....	16

7 Related Documents .....16

## 2 OVERVIEW

### 2.1 INTRODUCTION

The Administrative Case Event System (ACES) is an information system designed to facilitate the unification of Court Case information across many self-contained management systems which are each designated as the source of truth. When changes occur in a *Source System*, a command and event pattern is established to convey this information to other interested systems.

In order to expedite the transfer of information to a central point of reference, ACES provides a communication gateway with an immutable, “eventually consistent” model of data shared among systems. The structure of ACES is a Service Oriented Architecture.

Anne Thomas Manes describes it this way: “Although the word “SOA” is dead, the requirement for service-oriented architecture is stronger than ever. But perhaps that’s the challenge: The acronym got in the way. People forgot what SOA stands for. They were too wrapped up in silly technology debates (e.g., “what’s the best ESB?” or “WS-\* vs. REST”), and they missed the important stuff: architecture and services. Successful SOA (i.e., application re-architecture) requires disruption to the status quo. SOA is not simply a matter of deploying new technology and building service interfaces to existing applications; it requires redesign of the application portfolio. And it requires a massive shift in the way IT operates.”... “If you want spectacular gains, then you need to make a spectacular commitment to change.” (Thomas Manes, 2009)

These are the types of Architecture and Services we are discussing within ACES. The Services are asynchronous, durable, flexible and scalable. We leverage some current processes while re-engineering less optimal ones. Taking something that is designed to work locally and shoehorn it into a world of network calls will result in problems. Messages should be designed in a remote first mindset with explicit contracts that can support versioning and maintenance. We know demand always changes, for this fact we strive to accommodate growth and traffic patterns, while more importantly, rapidly adapting to changing needs of the business of the Courts. Court applications will use ACES to communicate with each other, with other agencies, and with the general public.

Historically, system integration has relied on the passing of flat files, synchronized central databases or point to point message delivery system like IBM Websphere MQ. The ACES approach provides notification of changes to any and all interested consumers with Event Stream Processing. It uses ideas from real-time data management, asynchronous scalability, and stateless messaging.

A great chasm often exists between Applications, Services and Content. There should be a direct path to what people want, when and where they want it. Applications must be smart enough to complete common actions without spelunking into a myriad of monolithic Applications in order to achieve it. ACES aims to bridge that chasm and provide the mechanism for elegant intelligence.

### 2.2 ASSUMPTIONS

It is assumed that the specific Features and Requirements for this system are not included in this document, but reside in a lower level Document targeting the specific Major Design Elements. Nomenclature with a defined meaning is highlighted with italics.

ACES is built as a Reactive, Event Based System: “Systems built as Reactive Systems are more flexible, loosely-coupled and scalable. This makes them easier to develop and amenable to change. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster.”<sup>1</sup>

The design approach of ACES is Distributed Domain-Driven Design — collaboration between development teams and business experts to produce useful models to solve problems. This commitment to collaboration and knowledge sharing must take place for development teams to gain the deep insights required to function within the problem domain. For a deeper understanding of Domain-Driven Design, we recommend reading Domain-Driven Design Distilled by Vaughn Vernon.<sup>2</sup>

<sup>1</sup> <http://www.reactivemanifesto.org/>

<sup>2</sup> <https://www.amazon.com/Domain-Driven-Design-Distilled-Vaughn-Vernon/dp/0134434420>

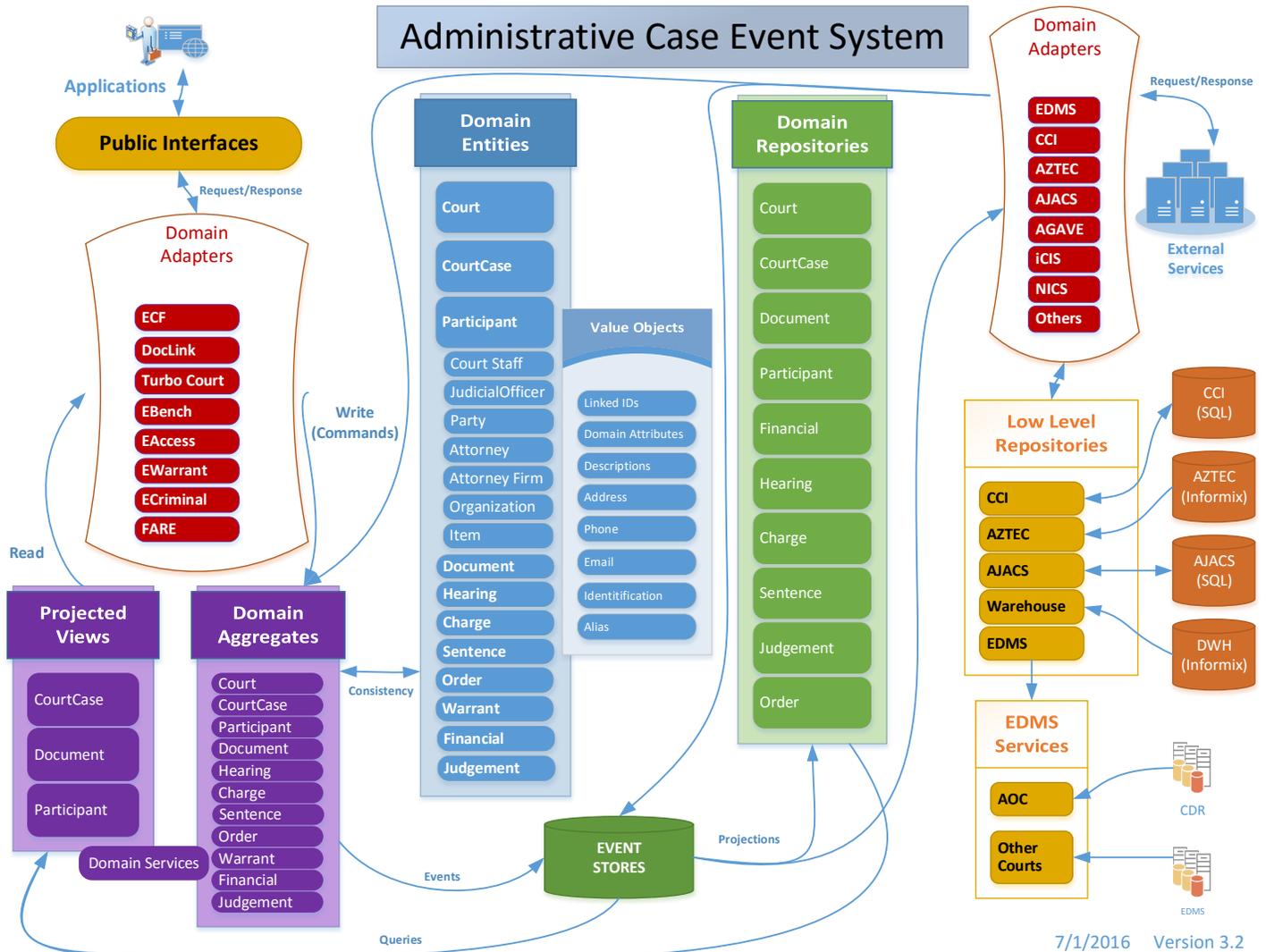
### 3 PROCESS DESIGN

The process design accounts for each Major Design Element using a well-defined and consistent methodology for accomplishing communication among parts of the whole system.

#### 3.1 PROCESS FLOW

The following diagram illustrates the process flow for integration of available data. It describes the paths that data and notifications take as they traverse the ACES system:

Figure 1



## 3.2 MAJOR DESIGN ELEMENTS

### 3.2.1 DESIGN GOALS

The driving goals behind the design of ACES is to use accepted standards, industry proven service orientation, and asynchronous messaging principles for the integration of Court Case related information.

The following list provides few Major Design Element goals:

- High Availability
- High Performance
- Resilient By Design
- Service Oriented
- Elastic and Decentralized
- Message based integration and interactions
- Extensible

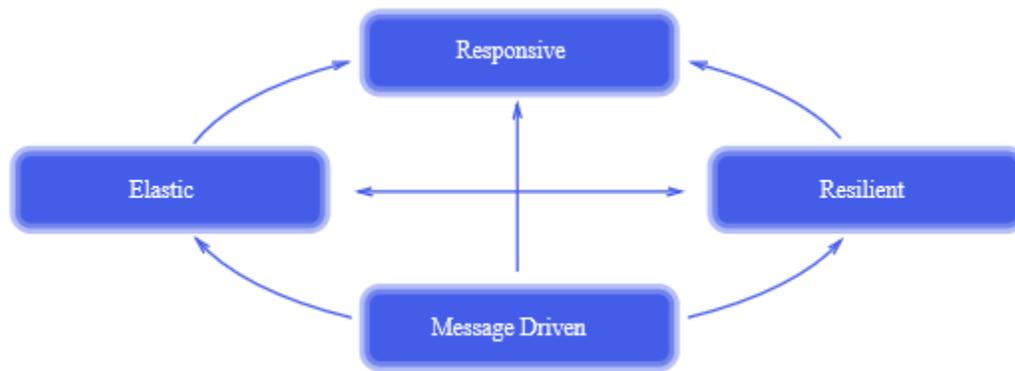


Figure 2- <http://www.reactivemanifesto.org/>

### 3.2.2 PUBLIC INTERFACE

Public Interfaces also known as API Gateways are how external Applications and Services interact with ACES. While these are typically Web Services, there are also integration points for IBM WebSphere Message Queues, Flat file consumption and Data Pulls. Collectively, these are referred to as the API. All Interaction is performed through a published set of Message Standards which are outside the scope of this document. See the Related Documents section for access to these external standards.

API Gateway resources are designed synthetically to follow client-driven use cases. When the client diverges from the *Canonical Message Model*, *Context Mapping* is used for translation. When working with MQ, a *Message Bridge* pattern is used.

Optimally, a connected application will communicate messages in real-time with ACES. This is the eventual intended methodology, though it is not the only available option. Legacy Flat Files and Timed Bulk Data Pulls are fully supported to allow for a lengthy transition period. Such legacy interaction still produces a set of Events that are routed the same way and become real-time bulk Messages. Routed Events provide the elastic scalability that is capable of processing large numbers of incoming Events by increasing capacity when needed and reducing capacity when no longer required.

---

### 3.2.2.1 VALIDATIONS

All Incoming Messages MUST be validated for authentication, authorization and content. Schema and structure validations will be provided using message contracts. These endpoints expose the Public Interfaces for ACES Integration.

Public Interfaces and Contracts conform to published standards which are outside the scope of this document, such as the Oasis ECF 4.01 Standard, the CCI Data Request Dictionary, and the DocLink Security Standard. See the Related Documents section for access to these external standards.

---

### 3.2.3 DOMAIN ADAPTER

Domain Adapters consist of *Message Endpoints*, *Message Routers* and *Message Translators* which adapt external systems into the *Canonical Message Model* of ACES. Domain Adapters are normally *Service Activators* in which the API Gateway delegates to an internal *Application Service*. This is being done now with eFiling, eWarrant, EDMS, AZTEC, DocLink, NICS and Interpreter Registry.

---

### 3.2.4 PROJECTED VIEW

Projected Views are *Document Messages* that are the response to a *Query Message* or delivered as the result of a *Domain Event* being handled. The 3 Projected Views depicted in Figure 1 are for illustration purposes only. There could be 1, 2 or a thousand different Projected Views for a CourtCase, Document or Participant, the illustration depicts that these Identities are the primary way of accessing the particular Projected View. As other Projected Views are required for optimizing specific applications, they are added to this section. These are usually referred to as “Data Transport Objects” (Fowler, 2003).

---

### 3.2.5 DOMAIN AGGREGATE

“Aggregates are a cluster of associated objects that we treat as a unit for the purpose of data changes.” (Evans, 2004)

Aggregates are what allow us to enforce business rules that must always remain consistent. It also allows us to optimize the flow of data to other external systems which may be listening for changes in CourtCase data.

This is not about changing or cleansing data, but rather how to notify an external system when ACES cannot use the data to communicate with another system as the data was presented to it. It also helps to optimize how we collect the correct information to send to an external system or to produce optimized Read Models for the data an Application needs.

Aggregates create Events — Success Events as well as Failure Events such as a consistency or communication failure. Events control and process the operation of Workflows and State Machines within ACES.

For example, Events may be used to alert a source system. Let’s look at a NICS example. When incoming CourtCase data is presented to ACES, it produces several Import Events. When a NICS Import Failure Event occurs, ACES can send a Message communication to the *Source System* alerting it that a localized change is required in order to have data participate in an exchange with NICS. This can happen in a specific way that is negotiated with the *Source System*. When the data is modified by the *Source System*, the Aggregate will then create the NICS Import Success Event which is collected by the NICS Event Handler.

When dealing with incoming data, the ACES system should not continually try to communicate data which it already knows is problematic. In the case of NICS, it does not collect anything but NICS Success Events. It does however, place the data in other Read Models which are compatible with the data for use with other systems. This strategy prevents incompatible data from continually being presented to external systems, while preserving it for use with other compatible systems.

This improves the resilience of the system as a whole and assigns the responsibility for correction back to the *Source System*.

---

### 3.2.6 DOMAIN ENTITY

An Entity in this material describes an object primarily defined by its identity and has a specified Lifecycle. Entities are typically a Value Object, or collection of Value Objects which are assigned an Identity within the system that is tracked and used to refer to the specific Entity thereafter. They have a beginning, a middle and an end which denote the Lifecycle. Lifecycles typically have special meaning on creation such as possibly assigning an Identifier. Additionally, upon removal, there may be special workflows which may need to be processed.

---

### 3.2.7 VALUE OBJECT

Value Objects are a collection of characteristics which describe something. They are changed together as a whole and when exchanging one for another there is no discernable difference to the system. While they may certainly be assigned an identity, the idea that they are interchangeable as a whole and remain immutable as a whole is more important.

---

### 3.2.8 DOMAIN SERVICES

Domain Services are operations which lie outside the scope of a specific Aggregate, Domain Entity or Value Object. They usually require more information than can be provided by those three patterns in an encapsulated manner. These services may act as *Pipes and Filters* for *Projected Views* or *Domain Aggregates* to restrict or enrich a *Document Message*.

One such scenario is the Security Model, this may use a Domain Service for routing and working with unauthenticated Users until they are authenticated.

Another scenario is the Business Rules that restrict Public Document Access, these rules operate as a filter for populating certain Read Models as well as providing filters for *Document Messages* through the *Projected Views*. The Rules governing one of the filters may be changed (outside the scope of a particular *Document Message*) and with that change, all functionality involving the use of this particular filter change at once. This alleviates any side-effects to other filters, rules, or processes which are treated separately.

One such service is the “Rule 123 Filter” which both populates the Read Model used for ROAM Indices used for Public Access as well as live Queries into the system for eFiling and DocLink. When Rules governing the “Rule 123 Filter” change, they change everywhere at once. Read Models using the filter can then be notified by ACES to rebuild themselves.

---

### 3.2.9 DOMAIN REPOSITORY

A Repository “mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.” (Fowler, 2003)

Domain Repositories are the internal data stores where Domain Aggregates and Projected Views are persisted. Domain Repositories are used internally by ACES as the sole place where *Domain Aggregates* persist data and from where *Projected Views* are populated. This allows for other *Read Models*, such as eBench, CCI and the NICS Repository to react to any Event which takes place within the Domain. *Domain Repositories* are responsible for routing *Events* related to incoming data changes and managing the transactional consistency of the *Domain Aggregates*.

An example use is when a bulk data pull is made from a remote Court. Court Case information is immediately persisted to the *Event Store* as an Incoming Data Event for each item of data. This Incoming Data Event is read and handled by the *CourtCase Aggregate* which persists the data in each *Domain Entity* to a *Domain Repository* within a *Transactional Boundary*. When the repository transaction succeeds, a new *Domain Event* is persisted in the *Event Store*, reflecting a change has been made to a *CourtCase Domain Aggregate*. Read Models subscribed to the *Domain Event* respond by persisting the change in the local Read Model format.

---

### 3.2.10 EVENT STORE

An Event Store is a specialized data store which records a transactional log of things that have already happened within ACES. Events are ALWAYS past tense and cannot be changed, they have already happened. A compensating Event can reverse a previous Event, but Events never mutate. Since these Events never change, it is an append-only repository and not intended for querying. Querying is supported through a business context specific Read Model.

Events may be published in order for multiple systems to react to the same Event. This would include publishing change events to multiple Read Models at the same time. Published Events for Persisted Data Models are known as Projections.

Once a Projection is available, Read Models subscribe to Events in that Projection and are populated with changes in near real-time to reflect the current state of its context. One distinct advantage of this Projection Model is that it allows for the inevitability that a Read Model needs to be taken offline. When the Read Model is brought back online it executes a catchup subscription to replay the Events it has missed since being offline. By replaying the missed Events, the Read Model is brought back into consistency by the existing Service, without a need for any special coding.

Since Read Models are updated from the Events, various data models can be configured differently from each other to allow for specific data optimizations and load characteristics according to the needs of the business context.

An additional benefit is that context specific Read Models can be added, extended or rebuilt directly from the Event Store by replaying Events without the need to re-query the original provider of the Event information. Events on specific Aggregates use a *Snapshot Model* to optimize the number of Events that need to be read to create an Aggregate or Projection. Snapshots also allow for older Events to be archived or removed after the snapshot is made.

“The Event Store uses a quorum based model for replication. This replication model ensures consistency throughout the replica group and is a well-known replication model. A main strength is that it is fully consistent. There are not possibilities of conflicting data on different nodes (i.e. A accepted a write without seeing the write to C). For most business systems this is a huge gain as dealing with these rare problems often gets looked over when using models that allows them.”

“Another strength is that failovers happen very quickly with minimal impact on clients. This is especially true when you consider that the nodes internally route so the client does not need to in most cases know who the leader is. Along with this, for a group of three nodes, so long as any two are up and communicating the system is considered running and consistent.”<sup>3</sup>

---

<sup>3</sup> <https://geteventstore.com/blog/20130301/ensuring-writes-multi-node-replication/>

---

### 3.2.11 LOW LEVEL REPOSITORY

Low Level Repositories are data stores that are the responsibility of the AOC and interact directly with the ACES Internal Services. While there are several other low level data stores which may integrate with ACES, those are accomplished strictly through Domain Adapters and an API Gateway. This separation allows extensible business rules, filters, routing and notification to occur throughout the system.

Currently there are 7 Low Level Repositories connected to ACES:

---

#### 3.2.11.1 CCI

CCI, the Central Case Index is a Read Model and is used for cross court rapid access to common Court Case Information. As Case Management Systems change Court Case related data, this is communicated to ACES which internally moves the appropriate change into CCI. Other systems may now access current statewide Court Case Information such as eFiling, eAccess and eBench. CCI also serves as a trigger point for the *Process Managers* that feed inter-agency notifications such as NICS. As other systems require notification, they are easily added as listeners which handle Events published with the ACES *Canonical Message Model* for processing. Several legacy systems that relied on the Data Warehouse for such processing are now being moved into ACES.

---

#### 3.2.11.2 AJACS

AJACS is the State Standard Case Management System. This is the Source Write Model for 13 Superior Courts and several Lower Jurisdiction Courts. More Lower Jurisdiction Courts are being added constantly until all AZTEC Courts have been moved to AJACS.

---

#### 3.2.11.3 AZTEC

AZTEC is the Legacy State Standard Case Management System currently being replaced by AJACS. During the time that AZTEC Courts are being moved to AJACS, AZTEC Data may participate in the ACES System.

---

#### 3.2.11.4 ROAM

ROAM is a Rapid Index system used by Appellation and eAccess for queries regarding Court Case Information. The ROAM System is being deprecated in favor of CCI for Appellation, though it is still being used as a front line Read Model for eAccess. This particular ROAM Index is however, populated by ACES.

---

#### 3.2.11.5 DATA WAREHOUSE

This Legacy Warehouse is a central repository for several legacy system's data. This is gradually being replaced by ACES.

---

#### 3.2.11.6 EDMS

EDMS is the Document Repository which consists of a Central Document Repository in OnBase as well as some local repositories for business unit specific applications. There is also a Message Gateway used to access remote Document Repositories which may deliver remote Documents on demand. For instance, when a Document from a Court that does not participate in the CDR is requested from ACES, it is delivered from the Message Gateway which retrieves it from the remote Court's provided Service Interface, which again are processed through a *Domain Adapter* bringing the result into the *Canonical Message Model*.

---

### 3.2.12 EXTERNAL ELEMENTS

External elements for which communication is necessary must use a Domain Adapter and Message Gateway to interact with ACES.

One of these elements is the National Instant Criminal Background Check System (NICS). When *Source System* (CMS) data is pulled, each incoming record issues a Command to Update a *CourtCase Aggregate*. The *CourtCase Aggregate* evaluates the incoming data for consistency and publishes *Incoming Data Events*. ACES Event Listeners (or “handlers”) react to the *Incoming Data Events*.

ACES Event Listeners produce other *Command and Event Messages* by applying Rules for various systems such as CCI and NICS. There is no practical limit to the number of ACES Event Listeners and they can be added or modified independently.

The ACES NICS Event Listener evaluates changes that qualify for NICS, when found, a *NICS Incoming Data Success Event* is produced. *NICS Incoming Data Success Events* are collected by a *Domain Service* reflecting the now current state of the CourtCase. The *Domain Service* issues a Command to send the data to the NICS System in its own format through the use of a *Domain Adapter*. NICS Responds to the ACES Command which produces a *NICS Data Sent Event*. This Event is then processed by *Pipes and Filters* to send the result back to the Source System (CMS). This may be accomplished by the *Pipes and Filters* sending the response as a *Document Message* to the *Source System* for notification. Alternatively, ACES could *Pipe* the response to a *Command* which invokes a *Composed Message Processor* responsible for composing an Email Report it will send at the end of the day.

Whatever the business context requires is achievable by ACES in this respect as a Reactive Event based system.

## 4 MESSAGE PROCESSING

“A *Message* is a fundamental building block when using message-based systems, one within which you package the information necessary to communicate actions and related data between systems.” (Vernon, 2016)

The Message functionality within ACES uses a technique that is called “Location Transparency.” Messages intended for a specific receiver can be communicated equally well whether the receiver is remote or local. Code is written without regard to where the receiver actually lives. Receivers may be in a local process, a remote process, or even a server in another building. The Code remains the same even if the receiver moves locations. Everything is designed to work in a distributed setting with information about the setting provided by configuration data. All interaction is purely message passing and asynchronous.

The asynchronous character of the system also helps to ensure the scalability of the system, so that all functionality is available equally when running within a single processor or on a cluster of hundreds of machines. In this respect, local communication becomes an optimization rather than attempting to expand later by adding remote capability.

ACES uses 4 Fundamental message patterns to control all information.

### 4.1 COMMAND MESSAGE

A Command is a request that causes a state transition (Woolf, 2004). A *Command Message* is a message intended to alter state within ACES, but may also elicit a *Response Message* usually in the form of a *Document Message*.

### 4.2 EVENT MESSAGE

An Event Message conveys information that has already happened with the system. Events Messages are used to convey state transitions to both internal and external systems. All Events within ACES are stored as transactional elements in an Event Store.

It is imperative that Events are verbs in the past tense, they are part of a Ubiquitous Language. In Domain-Driven Design terms, the Event makes the concept of what happened in the Event explicit, not something to be explored and defined.

### 4.3 DOCUMENT MESSAGE

Document Messages convey information without indicating how the data should be used. Quite often a Document Message within ACES can assist in managing workflow or long-running processes. As each step in the process completes, it may append to the *Document Message* until it is fully composed for delivery to the original requestor.

### 4.4 QUERY MESSAGE

A Query Message is a special kind of *Command Message* in which a reply is expected in the format of a *Document Message*. While the Command is not strictly requesting a change in state, it is requesting a reply that needs to be fulfilled. Fulfilling a Query may be done by an *Aggregator* or *Composed Message Processor* that compiles the results from several other *Document* or *Event Messages*.

## 4.5 COMMUNICATION MATRIX AND MESSAGE FORMAT

Messages internal to ACES use a *Canonical Message Model*. This *Canonical Message Model* is carefully tracked and versioned if any changes are necessary. Messages between ACES and External Systems will use a agreed upon Application Level messages through an API Gateway. Since these messages may not conform to the *Canonical Message Model*, the use of *Message Translators* for the external system *Messages* may be necessary.

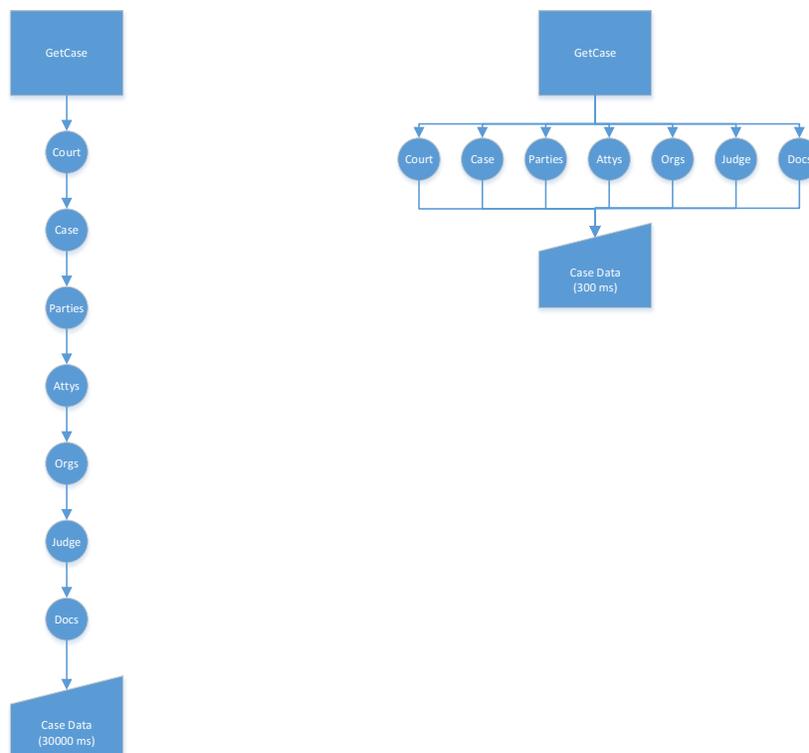
For Example, communication with the eFiling system is done through the use of an API Gateway, this external interface conforms to the Oasis ECF XML Messaging Standard. When a Message is received from the eFiling system it is routed to a Domain Adapter which invokes a Message Translator supporting the translation of an ECF XML Message into the *Canonical Message Model* which in turn executes *Command Messages* on ACES. This results in a *Canonical Document Message* being received in which a *Message Translator* translates the *Canonical Document Message* back into an ECF Formatted XML Message for the API Gateway to return to the eFiling system.

When the CourtCase Domain Aggregate assembles CourtCase Information from various Entities and Value Objects, the messages all conform to the *Canonical Message Model* of ACES and no translation is necessary.

Another example is, DocLink, a Public Interface which allows a authorized access to CourtCase Related Documents. Since this External Interface conforms to the *Canonical Message Model*, no translation is necessary. For accessing Documents not in the CDR, a *Message Translator* and *Domain Adapter* is used to talk to the remote EDMS which conveys the remote information back to the *Canonical Message Model*.

This methodology allows the *Canonical Message Model* to adapt to any external system with effort only being placed into a single Domain Adapter.

Messaging also provides an enormous gain in performance potential. Here I demonstrate how a synchronous messaging increases efficiency for eFiling. When requesting a CourtCase, we could process everything one after the other to build up the CourtCase, this takes 30 seconds, the sum of all the Queries... If I do the exact same request a synchronously in parallel, it takes only 0.3 seconds, the slowest of all the individual Queries.



## 5 DESIGN PATTERN REFERENCE

This section describes some of the Design Patterns used in this document.

### **Message Endpoint**

In a messaging system, the sender and receiver of Messages are both Message Endpoints.

### **API Gateway**

An API Gateway is a known location for external systems to interact with ACES through messaging.

### **Message Translator**

A Message Translator transforms the data from a Message to data that is compatible with the local, receiving application.

### **Message Router**

When a Message is received, some property of the message or the environment is checked and the Message is then routed to an appropriate Message Channel that satisfies the business or technical condition.

### **Pipes and Filters**

This is when you compose a process by chaining together any number of processing steps. Since these steps are not dependent on one another, they can be rearranged or replaced as the need arises. This forms a pipeline in which Messages are processed for delivery and the filter is what controls the direction the next step in the process takes. Order usually matters and rearranging the processing steps on the same message could alter the result.

### **Canonical Message Model**

A Canonical Message Model is different from Canonical Data Model. Coordinating stakeholders of every application to support a common data model is typically a futile effort that most often fails. On the other hand, a *Canonical Message Model* instead allows each application to define their own local models and still use a common set of Command, Event and Document Messages to convey information between systems. When systems elect to interact with ACES, exchanging information becomes a collaboration of how to compose and pass new or existing context specific Messages rather than realigning Canonical Data Models. This is a dramatically smaller effort than coordinating and catering to every stakeholder's specific data needs.

## 6 WORKS CITED

- Evans, E. (2004). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Boston: Addison-Wesley.
- Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley.
- Thomas Manes, A. (2009). *SOA is Dead; Long Live Services*. <http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html>: VP and Distinguished Analyst at Gartner, Inc.
- Vernon, V. (2013). *Implementing Domain-Driven Design*. Boston: Addison-Wesley.
- Vernon, V. (2016). *Reactive messaging patterns with the Actor model : applications and integration in Scala and Akka*. Boston: Addison-Wesley.
- Woolf, B. a. (2004). *Enterprise Integration Patterns*. Boston: Addison-Wesley.

## 7 RELATED DOCUMENTS

[Oasis ECF 4.01 Standard with AOC Extensions](#)

[CCI Data Request Dictionary](#)

DocLink Security Standard

[ACES Functional Requirements](#)