

**Project Report
CA-1**

Systems and Methods for Composable Analytics

**L.H. Fiedler
T.J. Dasey**

29 April 2014

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS



Prepared for ASD(R&E) under Air Force Contract FA8721-05-C-0002.

Approved for public release; distribution is unlimited.

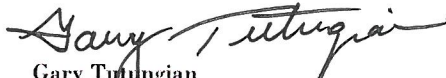
This report is based on studies performed at Lincoln Laboratory, a federally funded research and development center operated by Massachusetts Institute of Technology. This work was sponsored by the Assistant Secretary of Defense for Research and Engineering, ASD(R&E), under Air Force Contract FA8721-05-C-0002.

This report may be reproduced to satisfy needs of U.S. Government agencies.

The 66th Air Base Group Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER


Gary Tutungian
Administrative Contracting Officer
Enterprise Acquisition Division

Non-Lincoln Recipients

PLEASE DO NOT RETURN

Permission has been given to destroy this document when it is no longer needed.

**Massachusetts Institute of Technology
Lincoln Laboratory**

Systems and Methods for Composable Analytics

*L.H. Fiedler
T.J. Dasey
Group 45*

Project Report CA-1

29 April 2014

Approved for public release; distribution is unlimited.

Lexington

Massachusetts

This page intentionally left blank.

ABSTRACT

Composable Analytics is a web-based software platform that enables community researchers, analysts, and decision makers to collaboratively explore complex, information-based problems through the creation and use of customized analysis applications.

This page intentionally left blank.

TABLE OF CONTENTS

	Page
Abstract	iii
List of Illustrations	vii
1. MOTIVATION	1
2. APPROACH	3
3. TECHNICAL DETAILS	5
3.1 Application Model	5
3.2 Plugin Development	10
3.3 Module Type Implementation	12
3.4 Activation	13
3.5 Results	14
3.6 Discovery and Exploration	14
3.7 Boards	15
3.8 Groups	17
3.9 Security	17
3.10 Social Media	19
4. USABILITY	23
4.1 How-To Videos	23
4.2 Examples Applications	23
4.3 Descriptions	23
5. SOFTWARE ARCHITECTURE	25
5.1 Presentation Layer	26
5.2 Web Services	26
5.3 Data Layer	26

TABLE OF CONTENTS (CONTINUED)

	Page
6. LEN PREPARATION	29
6.1 Fixing Security Vulnerabilities	29
6.2 CDC Morbidity and Mortality Weekly Report Data	29
7. FUTURE WORK	31
7.1 Commercialization	31
7.2 Technical Features	31
8. FINAL THOUGHTS	35
APPENDIX A. LIST OF MODULES	37

LIST OF ILLUSTRATIONS

Figure No.		Page
1	Core system entities and their relationships.	3
2	Data integration platform.	4
3	Application in designer with intermediate results.	6
4	Application designer showing a run-time error.	8
5	Simplistic module.	11
6	Simple web service created in designer.	14
7	Discover page showing application previews.	15
8	Results and applications organized on a board.	16
9	Example access control list for an application.	19
10	Options for linking social media accounts.	20
11	Application landing page.	21
12	Three-tier architecture.	25
13	Entity relational diagram.	27
14	Technologies used.	28

This page intentionally left blank.

1. MOTIVATION

Today's analysts gather more data than ever before. More importantly, analysts have or wish to have access to a considerable number of disparate data sets. There are many questions surrounding the nature of integrating and analyzing data; users need a flexible environment to investigate data, and to easily explore new techniques. Furthermore, most analysts do not examine the same data every day, nor are they continually using the same methods. Their workloads are variable, and require adaptability over short periods of time.

When creating a system to process data, the most common approach involves hiring a team of developers to automate the reading of data and send it through various analytical methods. Obvious shortcomings to this method include cost and flexibility. Each change an analyst wants to make requires modifications by the developers. Opportunities are lost when software developers cannot work quickly enough to meet event-driven analyst needs, or when the cost of developing custom software is prohibitive.

Users and analysts have a wide variety of issues and characteristics that can be overcome through end-user development techniques, such as Composable Analytics. Analysts typically do not have the development skills necessary to develop complete end-to-end software solutions for their work. They are typically under a considerable amount of time pressure and have limited software development budgets, which can result in a reduction in analysts' effectiveness. If we were to provide end-user development techniques to shift the development of new capabilities onto the user, enable collaboration aides, and allow for rapid construction of applications, then users could be in much better position to accomplish their tasks.

Furthermore, current software is not capable of meeting analysts' necessary goals when unanticipated needs and issues arise, typically at the worst time. To add to these complexities, data is typically distributed and in an unfamiliar format and ontology, causing issues in pulling all the necessary data together to answer the question at hand. Providing ways for users to create new data processing schemes and visualizations allows them to be much more productive.

Users' domains and functional areas within those domains are variable. While analysts are typically trying to solve domain-specific problems, at the end of the day, they are all viewing and manipulating data. It would be beneficial to provide a core data and integration platform, and allow users to plugin in their domain specific functions.

A growing population considers themselves "programmers." While they use this term loosely, they are creating new software functionality. Some of this functionality includes websites, interactive visualizations, and online surveys. In fact, Gartner predicts that more than 25% of business applications

will be developed by “citizen developers” in 2014.¹ People no longer want to be passive users of software, but active participants in the development cycle. Users no longer want to simply create data artifacts, they desire to create functional software artifacts.

¹ <http://www.gartner.com/newsroom/id/1744514>

2. APPROACH

The power of adaptability should extend beyond the realm of software developers and into the capacity of analysts and users.

Composable Analytics is a web-based environment for users to explore, create, and collaborate on analytical methods. We recognize collaboration is a key factor in the way users create methods and understand their results. In the system, users can explore and find existing applications that others have created. If an application meets their needs, they can use it wholesale. However, if their requirements are slightly different, they can either reuse the application inside another, or they can clone the application and modify it for larger software behavioral changes. Once users run their applications and produce results, they can begin to organize their results in boards. Boards are a way for users to reference results, applications, and other information, and share it with colleagues in a meaningful way.

There are five core entities within the Composable Analytics environment: users, groups, applications, boards, and results/runs. All entities are considered securable resources and have various access control permissions that can be applied to the individual resources. As users begin to explore and use the system, we anticipate groups forming around common interests and social structures. Groups can also serve as a way of managing permissions for a large number of users. Users are at the root of the system, and we anticipate environments initially being seeded by a small number of users creating applications. As results are formed, and boards are created, users will have content to review and modify. Seedling applications will begin to be cloned, modified, and run with new datasets.

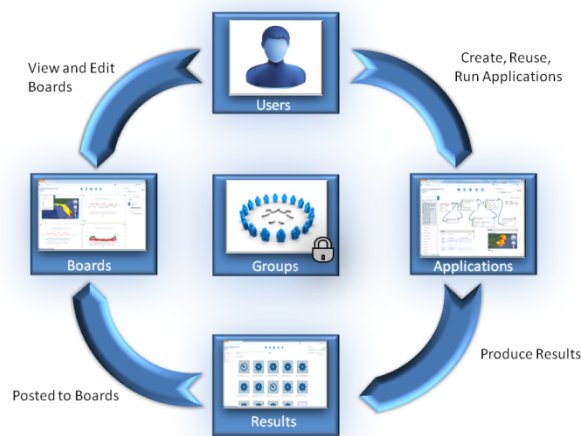


Figure 1. Core system entities and their relationships.

Composable Analytics can be viewed as a data and process integration platform. We live in an extremely chaotic data environment. Data exists everywhere in different formats, and analysts have access to tremendous amounts of it; yet, most data is silo'd in individual systems, and these systems typically have different tools for viewing the data. When needs arise for cross-cutting analysis or for analysis methods not supplied by the individual systems, the data is typically downloaded and analyzed offline in tools such as MATLAB, R, or Excel. Furthermore, if there's a requirement to develop software to integrate the data sets, typically the analysis will not be performed because the software cannot be developed quickly enough, or a new data strategy will be created and a large development effort is started, resulting in lag time and substantial cost.

Composable Analytics allows users to query many information sources in a variety of formats and integrate them using a flow-based application. Entire datasets do not need to be pulled; only aggregated or selected data is transferred. Processes and queries can be sent to other data systems, rather than the data being sent to the process. Existing analysis code can also be reused and integrated – whether it is written in MATLAB, Excel, or other languages. Composable Analytics separates the method from the data, allowing users to simply share and reuse applications, and visually display applications and data in a cohesive manner.

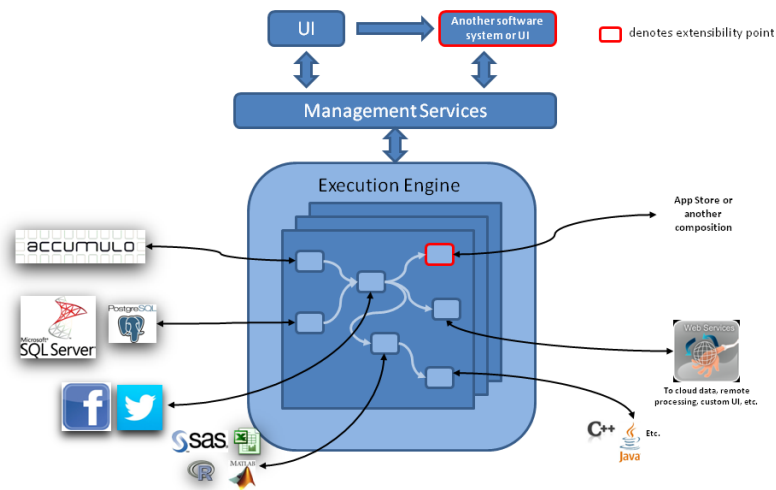


Figure 2. Data integration platform.

3. TECHNICAL DETAILS

3.1 APPLICATION MODEL

3.1.1 Overview

An application within Composable Analytics consists of functional blocks strung together with connections to create new functionality. These functional blocks, called modules, have multiple inputs and outputs. A module is analogous to a function in any programming language. A module takes in one or many inputs, and produces one or many outputs. These outputs can then be connected to any number of other module inputs.

An application can also reference and call another application. This reference is just like any other module, but its functionality is to call the referenced application. To provide inputs into the application, and retrieve certain outputs, the referenced application developer needs to mark certain module inputs and outputs as externalized with additional data (name and description). The user does this by adding special modules that externalize inputs and outputs. These external modules then become the inputs and outputs in the ‘application reference module’ in the calling application.

Our application model is very similar to dataflow models. At its roots, the model is a directed graph, where the nodes are the modules, and the edges are the connections between module inputs and outputs.

Each input and output on a module has a name, description and a type. Inputs and outputs are strongly typed and we are leveraging the .NET framework type system. Objects of the same type can be assigned, and objects having types that extend base types can be assigned to inputs of the base types. For example, outputs of types ‘integer’ can be fed into a module input of type ‘object.’ We also allow convertibility. For example, if a user connects an output of type ‘string’ to an input of type ‘integer,’ then we try and do the conversion for them automatically. Note that this can result in exceptions during the running of the application. We are currently using the built-in conversion framework in our type system, and new input and output types can register conversions by creating casting operator.

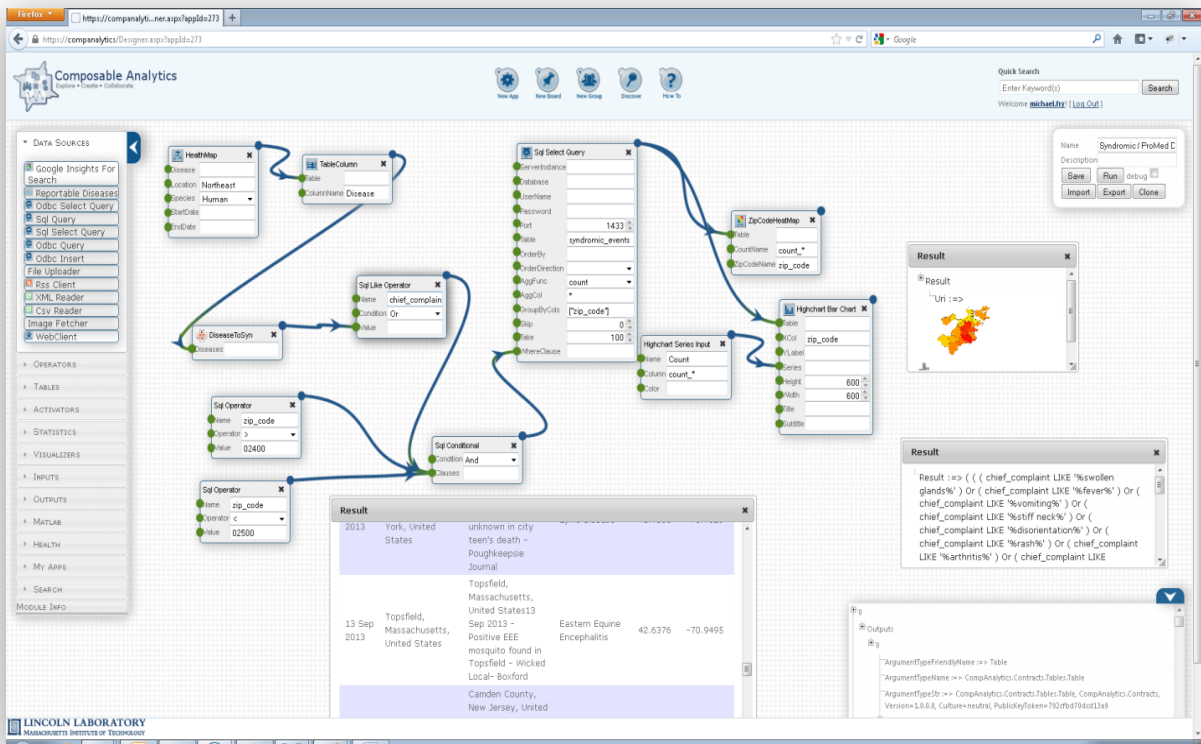


Figure 3. Application in designer with intermediate results.

3.1.2 Running of Applications

The most common way of running an application is in the designer. As users develop an application, they can run it at any time by clicking the run button. Because an application is fundamentally a directed graph, the system knows the order in which modules need to be executed (there can be many valid orders). The system then loops through this order and execute the modules. Before each module execution, the system assigns the inputs to the module based on the input connections.

Although the user creates and interacts with the application in a web browser, the actual execution of the application occurs in the backend web servers and status and results are sent back and forth in a real-time manner.² Whenever the application is run from the user interface (UI), the entire application

² Note that if a module in an application is actually a web service, it can also result in computations or queries on remote systems outside the Composable Analytics server.

model in the browser is sent to the application web service for execution. This is required because the user may want to change the application and run it, without saving the application to the database. Designing and running an application within the same space creates a very interactive experience for the user.

3.1.3 Debugging Applications

We have developed a debugger to aide in troubleshooting the creation and running of applications. Users can step through an application run, halting execution after each module execution. This allows users to understand the order in which modules are running and also look at intermediate outputs. If an application is calling another application, the user can ‘step into’ and start debugging the referenced application run. Assuming they have read and execute permissions, the referenced app and the current execution context is loaded in another designer tab, where they can then step through and debug the application just like the parent application. Once complete, they can then switch back to the parent application to continue debugging the current run.

If the user has enabled debugging, the execution context is set with a debug flag, and the application executor on the server will wait for a “step” call from the user before continuing execution. If the engine is stepping over a module, the call step returns the results of that module, and also notes which module will be executed next. This allows the UI to let the user know which modules have been executed, and also which one will be executed next. If the module that is executing is a reference to another application, the step call returns handle information, giving the UI the ability to launch the designer loaded with the nest application in debug mode. The calling application will then wait until the nested app is complete, similar to other module step calls.

3.1.4 Error Handling

If an error occurs during an application run, an exception is thrown and the run is stopped. If a module throws the error (from validating inputs, connection errors, or timeouts), then we bookmark which module threw the exception, and display the error on the correct module in the designer. In addition, if the exception is due to the assignment of a module output to an input, we also bookmark this information and highlight the input and connection in the designer for the user. This allows users to understand and resolve the root causes of application errors.

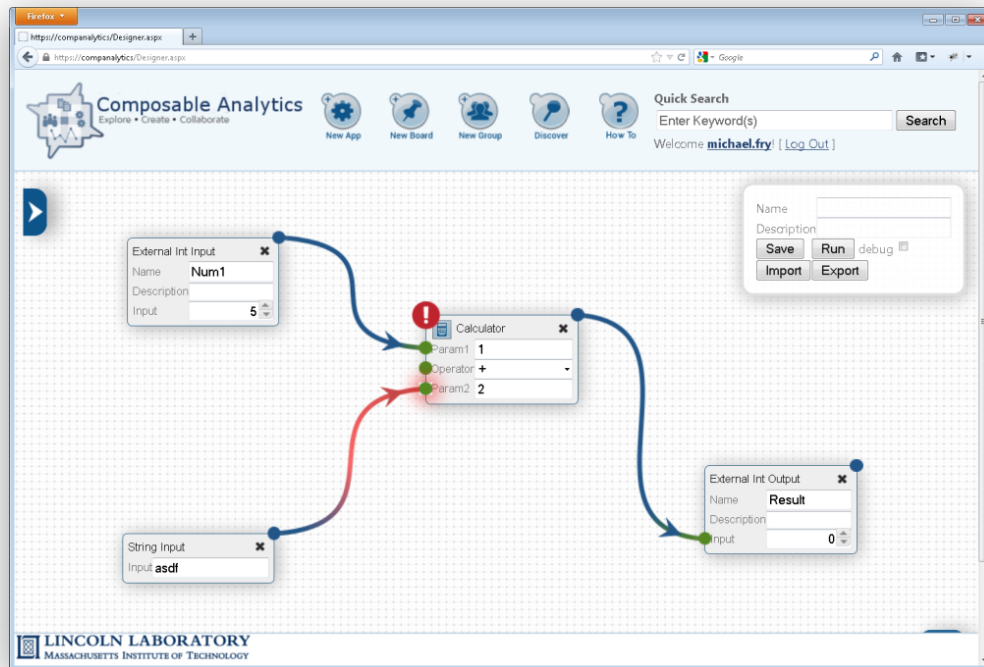


Figure 4. Application designer showing where a run-time error occurred.

3.1.5 Code Modules

3.1.5.1 Sandboxing User Code

We cannot predict and develop all the modules users will ever need. However, we strive to make inserting new functionality easy, and provide various tiers of flexibility. One of these tiers allows users to employ a ‘just-in-time coding’ methodology. In addition, we do not want to allow any user to develop a module and have the system trust its execution. To provide this flexibility, and also meet our security requirements, we have developed a ‘code module.’ Like any other module, it has inputs and outputs. However, the input to this module is user code, which can be written in C# or VB.NET. The user code is compiled and then run in a partial trust application domain, also known as a sandbox. Users are not able to access the file system, registry, or network. They also have limited memory and runtime constraints. Note that because the code is actually an input into the module, a user can potentially create some very dynamic programs. Applications can now receive code from external sources like emails or web services, and even modify the code based on other factors while the application is running.

When using a code module, the user specifies the language, class name, and method that should be called in the sandbox. The module also takes in a collection of inputs. These inputs correspond to the parameters on the method. For example, if the method took in three parameters (string, Table, integer), then the application designer would link up three connections to the method input collection. The output of the module is of type object. This corresponds to the return value of the method called in code. If the method returns a void, then null is returned.

3.1.5.2 Execute MATLAB Code

We also currently support the execution of MATLAB code. There are, however, several differences in the way we execute MATLAB code versus sandboxing .NET code. We use MATLAB's automation server to host an instance of the MATLAB runtime, and use the automation server's application programming interfaces (APIs) for creating workspaces, executing commands, and retrieving and setting variables in the runtime. Currently, there is no way to sandbox and restrict the execution of MATLAB code. This opens serious security vulnerabilities. To mitigate the security issues, we execute the automation server under a limited user account. However, even under a limited Windows user account, there are still many serious operations the caller can execute that we would like to prevent (i.e., directory browsing, creating files). Therefore, execution of the MATLAB module is currently only supported for certain user accounts.³

Each run of an application that contains a MATLAB module creates a new workspace in the MATLAB automation server. This provides isolation from other application runs. MATLAB code can be a series of commands that are interconnected through sharing state within the workspace. For this reason, execution is slightly different from the .NET code module. There are three modules that the application designer uses to interact with MATLAB. The first is the "Put Variable" module. This module will put a value from the executing application into the current MATLAB workspace. Data with atomic types (integer, string, double, etc.) and multidimensional arrays are supported. The "MATLAB Code" module will execute the specified command in the current workspace. Finally, to retrieve values from the workspace, the designer can use the "Get Variable" module that will retrieve a variable from the workspace, and set the module's single output value using the MATLAB variable.

³ Note that this brings up a small, but required design change. Module types are not currently securable resources, and can be executed by anyone. If we want to restrict users from executing certain module types, module types should be promoted to securable resources – just like applications, groups, and boards. This will allow administrators to change the permissions on an individual module type. Permissions on a module type may include: Discover, Execute, Write.

3.2 PLUGIN DEVELOPMENT

The palette of modules the user can choose from dictates the kinds of applications that can be created and also steers the information domain and use of Composable Analytics. Therefore, we set off to make it very easy for a developer to quickly add new modules to the system. An ‘App Store’ concept is a possible avenue we can take to include new visualizations and first-class modules into the system. The business case and revenue stream for developers to submit new modules would need review based on the application area needs and constraints.

3.2.1 Module Type

Creating a new module type is the primary extension point a developer can make use of in the environment. The module type defines its name, description, icon, category, inputs, outputs, metadata on the inputs and outputs, and the actual class that will be executed when an application is running. These ‘first class’ modules can be written in any CIL (Common Intermediate Language) compliant language. Some of these languages include: C#, C++, VB.NET, IronPython.⁴

Below is a very simplistic module that performs a mathematical operation on two numbers. The most important method is the Execute() method. This will get called when it is the module’s turn to run. Within the Execute() method, the module can get access to the values of its inputs (whether the values are coming from connections, or set directly on the inputs), do any processing it needs, and then sets its output values. The inputs and outputs are set as properties on the class. ModuleInput<> and ModuleOutput<> are generic classes, and the use of these determines the type of data they will accept or return. In addition, an input control is also specified in the example below. In this example, the mathematical operator can only be chosen from a preconfigured list through a combo box. By adding a control attribute to the input, the UI knows to use the specified control when the user wants to enter in the input.

⁴ A full list of CIL languages can be viewed here: http://en.wikipedia.org/wiki/List_of_CLI_languages

```

[ModuleType(Name = "Calculator", Namespace = "edu.mit.ll.companalytics", Category = ModuleCategory.Operator, Icon =
"./images/module-icons/calculator.png")]
[Description("Performs numerical operation on two numbers")]
public class CalculatorModuleExecutor : ModuleExecutor
{
    [Description("First numerical input")]
    public ModuleInput<double> Param1 { get; set; }

    [ComboBoxControl("*, +, -, /, %", "^")]
    [Description("Numerical operation")]
    public ModuleInput<string> Operator { get; set; }

    [Description("Second numerical input")]
    public ModuleInput<double> Param2 { get; set; }

    [Description("Output of operation")]
    public ModuleOutput<double> Result { get; set; }

    public override Module GetDefaultModule()
    {
        Module module = base.GetDefaultModule();
        module.ModuleInputs.First(m => m.Name == "Param1").ValueObj = (double)1;
        module.ModuleInputs.First(m => m.Name == "Operator").ValueObj = "+";
        module.ModuleInputs.First(m => m.Name == "Param2").ValueObj = (double)2;
        return module;
    }

    public override void Execute(ExecutionContext context)
    {
        double param1 = this.Param1.Get(context);
        string op = this.Operator.Get(context);
        double param2 = this.Param2.Get(context);

        double result = this.Operate(param1, param2, op);
        this.Result.Set(context, result);
    }

    double Operate(double val1, double val2, string op)
    {
        switch (op)
        {
            case "*":
                return val1 * val2;
            case "+":
                return val1 + val2;
            case "-":
                return val1 - val2;
            case "/":
                return val1 / val2;
            case "%":
                return val1 % val2;
            case "^":
                return (double)((int)val1 ^ (int)val2);
            default:
                throw new NotSupportedException(String.Format("Operator {0} is not supported", op));
        }
    }
}

```

Figure 5. Simplistic module.

3.3 MODULE TYPE IMPLEMENTATION

3.3.1 Types

When defining a module type, the data types for each of the module's inputs and outputs are specified in the property declaration. These types determine which information can be sent between modules, and defines how modules interact with each other. Module type developers can simply create new data types by creating a new class and then using that class when defining a new module input or output. These types must be serializable so the information can be sent to the UI, and also saved to the database. Because these types are known at compile time, registration and loading of the types is inherent in the .NET framework.

3.3.2 Input Controls

Input controls help users set the value of a module's input. Not all values can come from connections, so the module developer should provide an easy way to choose values. Developers can register new input controls into the system. Module developers can make use of these by adding the input annotation to the input property in the module type class. When creating a new input control, there are several components that need to be created.

Registering Attribute – This is how module developers make use of the control and let the system know to use the control type on the specified input. Any options on the UI control are also typically specified here. In the example above, the module developer used a `ComboBoxControl` attribute so users could choose the operator from a dropdown. The attribute also specifies the JavaScript file containing the user interaction methods.

Data Parameters Class – Any optional parameters need to be serialized and sent from the services to the UI when loading the input controls. Any parameters set in the attribute are placed in this object and marshaled to the UI.

Java Script Control – On the UI side, the input control extends from a base `InputControl` class and must implement several methods. Some of these include operations for setting the input value, rendering, and getting the value from the control.

3.3.3 Output Rendering

Developers can also register how certain output types are rendered. Unlike input controls, which are defined at the module input level, output rendering affects all module outputs with the same type. From our experience, inputs typically need different ways of getting data from the user, even if the type is the same. For example, if the input is of type string, some modules may need a dropdown list of predefined values, while others may want a textbox. However, if the result is a string, then typically a textbox displaying the string is sufficient.

3.4 ACTIVATION

Applications do not always have to be run by the user. Users can design applications to be triggered by other external factors including: time, email, and web requests. If a user wants to run an application at the same time every day, he can add a timer module to his application and set the specific schedule. We have an activation service running in the backend looking for applications with timer modules, and checking whether it is time to run the application.

We also have a server periodically checking a system email inbox for application emails. If an email is received with an application ID, the message contents are assigned to the email module inputs in the application, and the application is run.

An application can also act like a web service or a web page. Users can simply drag a ‘web receive’ and a ‘web send’ module into their application. Next, users fill in the logic of processing the request and assigning the response with more modules and connections. The web request activator running in the backend receives web requests, and determines which application needs to run based on the specified application ID in the request. The service then assigns the web request contents to the web receive module and runs the application. The activator then takes the results from the application and assigns them to the web response. Users cannot only coordinate calls to web services, but now they can graphically program the internal workings of web services. Callers of the application web service need execute permissions on the application. The web request for the application should have a correct authentication cookie, or if everyone has been given ‘execute’ permissions on the application, then an anonymous web request will be accepted.

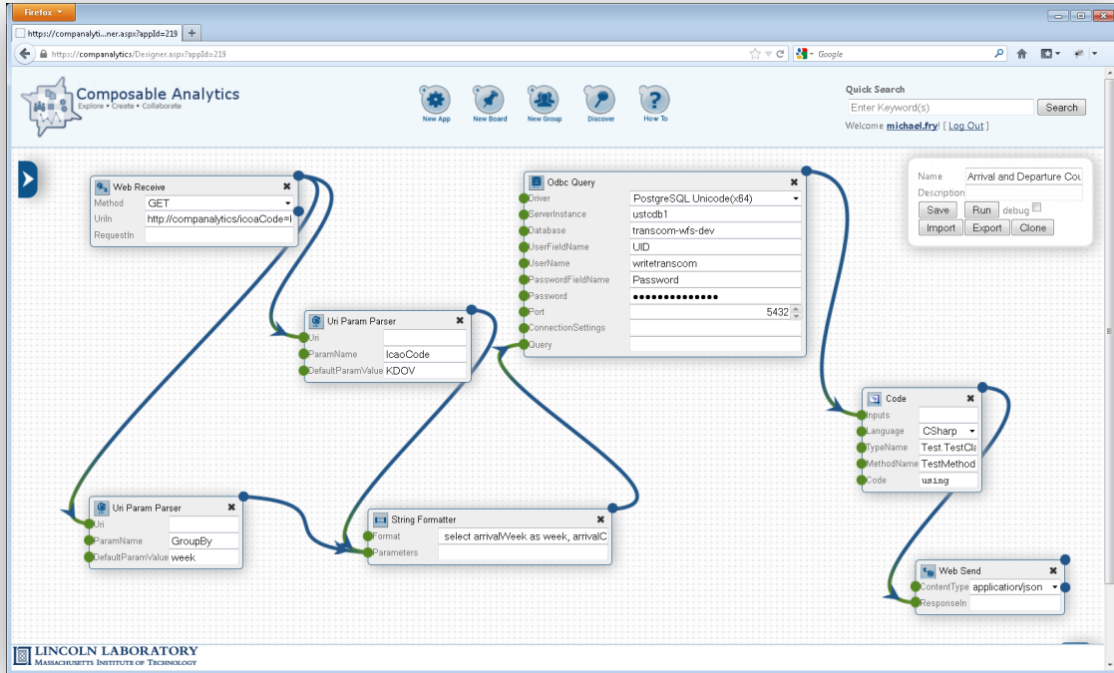


Figure 6. Simple web service created in designer.

3.5 RESULTS

As applications are executed by users or through external activation, runs are produced. A run consists of results for each module's outputs. Results are strongly typed objects, and correspond to the output of a module. Examples of results include graphs, charts, maps, strings, integers, and tables.

Runs and results are also securable resources, similar to applications, boards, etc. Currently, a run and its included results share the same access control entry. This is something that could change in the future if users feel the need for finer grain sharing on their results.

3.6 DISCOVERY AND EXPLORATION

Reusability is a key tenant to Composable Analytics. And to promote reusability, users need a way to find existing applications. We have developed a search capability for users to find the five main resources in our system: users, applications, boards, groups, and results. Users can search by keywords and specific attributes on the entity. We are using a full text indexer on the names and descriptions of

entities. For example, a user could search for applications that contain ‘tornados’ in the description and the structure of the application contains a visualization module of type ‘line chart.’ Also note that users only see search results of entities that have discover permissions; results are an intersection of key words, attributes, and permissions.

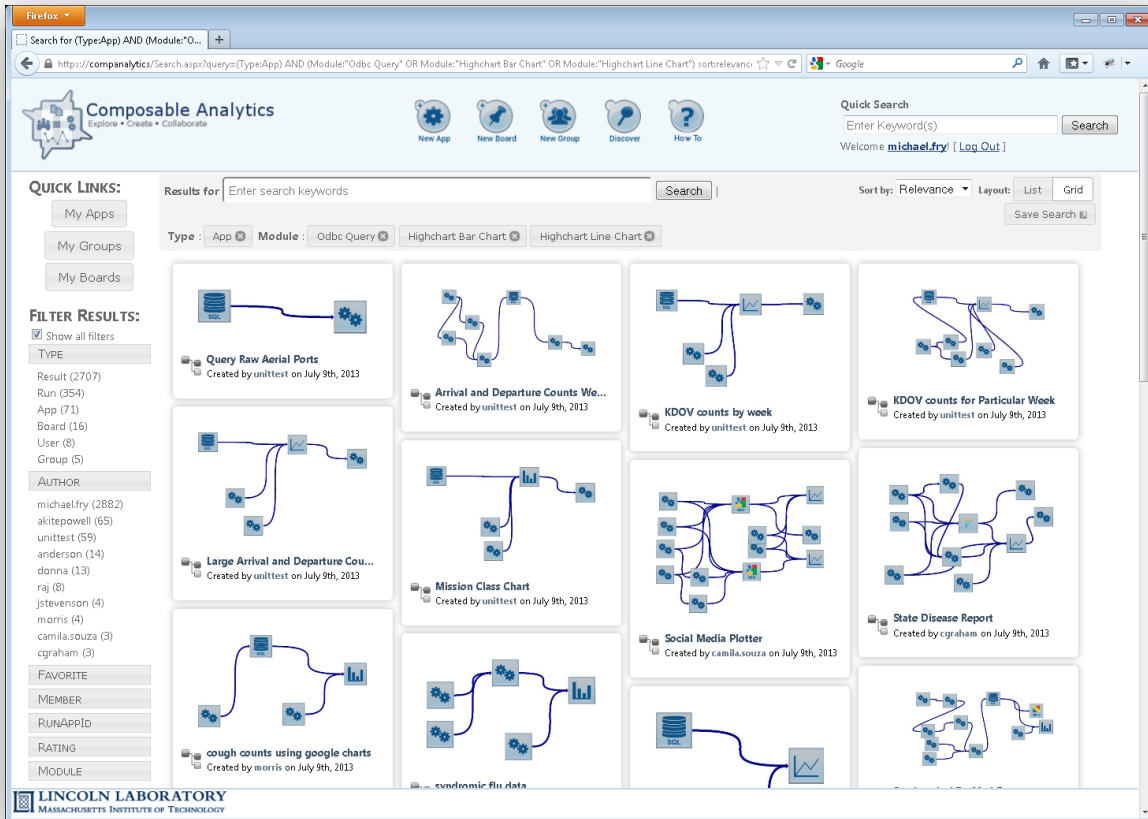


Figure 7. Discover page showing application previews.

3.7 BOARDS

3.7.1 Overview

Boards give users a way to organize application results and information in a meaningful way. For example, the output of an application may be a graph, but a lone graph does not have any context or anyway to compare itself with other graphs. To resolve this issue, users can pin the graph to their board,

drag it to the appropriate location, write text describing the result, and even compare it to other graphs on the board. Boards are similar to wikis or web page development where they give users flexibility to develop a view of their data. However, boards are purely visual. Users do not need to know a wiki language, or know HTML. They simply drag, resize, write text, and adjust attributes like color and font size of their objects.

Boards can be shared and multiple users can contribute results to a particular board. Boards can be automatically updated by applications publishing results to a board. We anticipate communities forming around boards, consisting of users that either want to contribute or stay informed about a particular information domain.

Boards contain a list of board items. A board item is associated with a resource (application, result, group, etc.) and also contains some additional metadata about the visualization of the resource. Some metadata attributes for a board item include: name, color, font, size, and position.

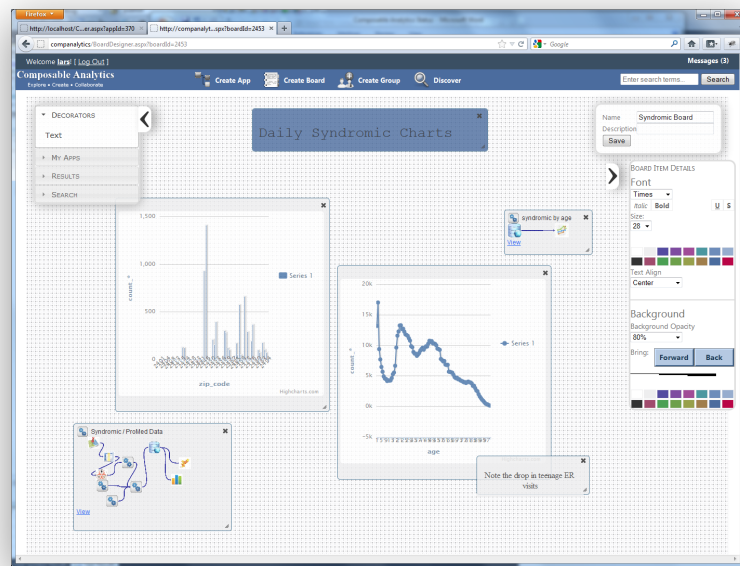


Figure 8. Results and applications organized on a board.

3.7.1.1 Live Elements

As resources are created, changed, and used, this information is recorded into an activities table. Users may want a quick way to know what actions/activities have been taken on resources. For example,

if an application has been updated and run, the activity log will show what user has updated the application, and also who has executed it recently. These lists can be pinned on an individual's board, and will be updated whenever changes occur. In addition, queries for resources in the system can also be placed on the board. For example, if the user typically wants see new applications that have been created in the system, they can create a query and view a live element list of the results on the board.

3.7.1.2 Resources

All types of resources can be added to the board. The most commonly added resource is a result (chart, map, or table). However, other resources (applications, groups, and other boards) can also be added. Previews of the resources are visible, allowing users to quickly jump to and make use of the resource.

3.7.2 Main Dashboard

Main dashboards actually function and are stored exactly like any other board in the system; however, when a user logs on, the user is redirected to the main dashboard. Individuals can customize their own dashboard. In addition, while dashboards can be shared like any other board, we anticipate only the owners having access. By default, the main dashboard displays live elements for favorite boards, groups, and applications. Other default live elements include newly created and highest rated applications. While users can post results to the dashboard, we anticipate the main dashboard being used as a quick link to resources.

3.8 GROUPS

Groups are another type of resource in the system and serve two primary purposes. Groups first provide a collaborative mechanism, allowing users of common interests to find each other. In addition, groups aide in securing other resources within the system. Rather than giving permissions at the individual user level, permissions can be specified at the group level. For example, if you want to give only certain people write permissions to an application, you can create a group, assign the group write permissions to the application, and then add users to the group.

Users can be added to the group by the owner, and also by any user or any other group, with write access to the group. Users can also request to join a group. Group owners are sent an email when a join request is created, and owners have the option to accept or reject it.

3.9 SECURITY

We have gone to great lengths in securing Composable Analytics, and security is a key aspect in all of our design decisions.

There are several security infrastructure mechanisms currently in place. At a transport level, all web traffic is encrypted and sent over SSL (Secure-Socket-Layer). Web server and activation processes run as

limited privilege accounts (NT Authority/Network Service). Processes accessing data use accounts with the minimum rights necessary for retrieving and saving the data.

Application designers are provided a considerable amount of power when using modules. For example, they can issue web requests to remote web services, run queries on external databases, run custom code modules, and automatically send emails. We have worked to not only protect the system from malicious users, but also to protect users from one another. At a system level, employing a web proxy and spam filters can mitigate some malicious use cases, but resource quotas must also be implemented in the future. We plan to address system resource quotas as our user-base grows. Granting users the tools to secure their applications is also important. Users currently have the ability to specify granular permissions on applications (i.e., who can execute, write, and read private variables).

3.9.1 Accounts

Users create accounts in a similar fashion to any other website.

The typical registration workflow steps include:

- User creates an account through a web form
- Email is sent to user ensuring a valid email exists.
- The user clicks on a verification link in the email.
- An email is sent to an authoritative person to accept or reject pending the account.
- If the account is accepted, then the account is enabled.
- Email is sent to the user letting them know the account is active.

Some email addresses can be auto-approved. If an email address domain (i.e., ll.mit.edu) is white-listed, then an authoritative person does not need to review the request. Once the account has been verified, it is automatically enabled.

3.9.2 Resource Permissions

While one of Composable Analytics' key tenants is to promote sharing, we recognize that users still need the ability to have fine-grained permissions on their applications and data. By default, any resource created by a user can only be seen by that user. A user can assign either another user or group particular permissions to an entity. For an application, the permissions are 'discover,' 'read,' 'read-private,' 'write,' 'execute,' and 'clone.' For boards and groups, the permissions are 'discover,' 'read,' and 'write.' Note the circular dependency between groups and permissions. This allows users to create interesting permission

functionality like allowing individuals within a group to see who else is in the group. This functionality can be achieved by giving the group read access to itself.

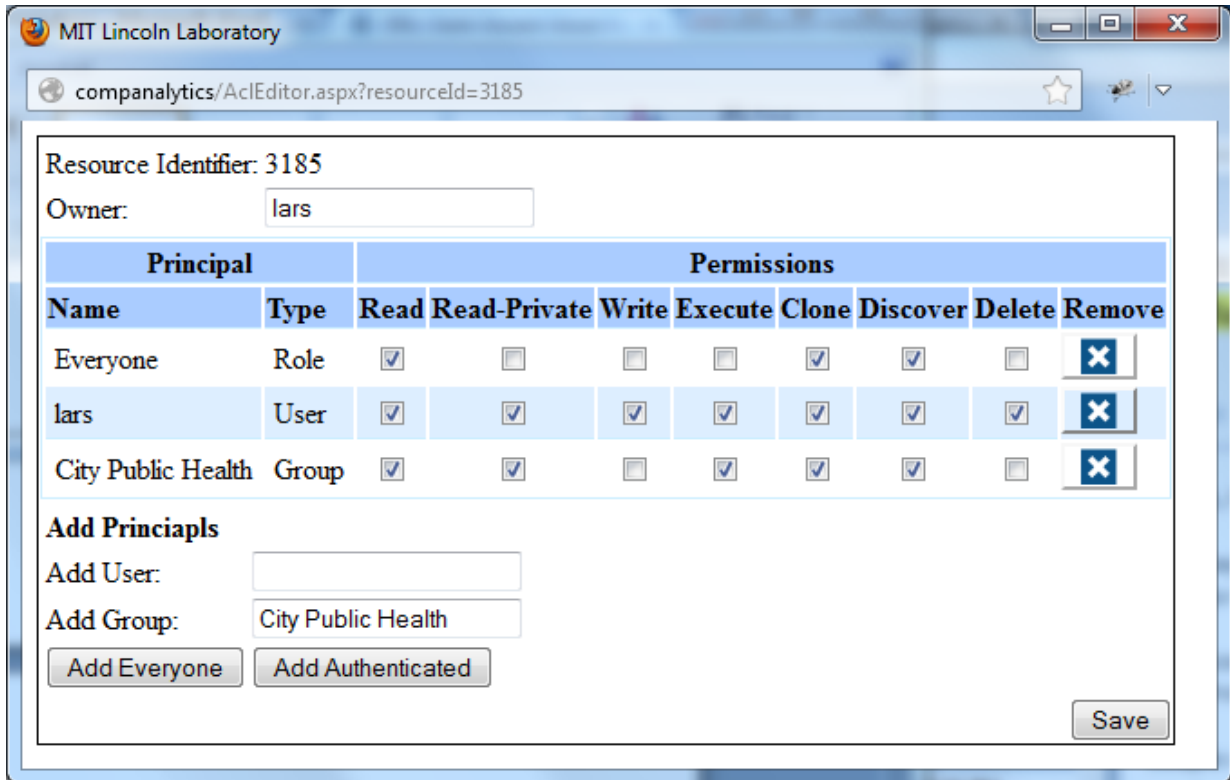


Figure 9. Example access control list for an application.


3.10 SOCIAL MEDIA

We recognize that most people want to share their work. In light of this, we have integrated with Facebook and Twitter. Users can link their Facebook and Twitter accounts to Composable Analytics using the Oauth protocols, and give Composable Analytics permission to post on their behalf. We have Facebook and Twitter modules that allow people to automatically publish either text or images to their feeds. Users can now automate and script daily public status updates.

Composable Analytics can also be used to automate social web applications. Users can create processing schemes to automate the process of posting to Facebook and other social sites. For example, if

a company has an approval process for posting daily updates to a social site, the workflow can be orchestrated within the Composable Analytics environment.

Connect your facebook account:
Lars Fiedler uses Composable Analytics.



Currently Linked User: **Lars Fiedler**
Expires On: **Friday, October 05, 2012 1:35:39 PM**

Additional Options

Publish Actions	App	Board	Group
Create	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Update	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Run	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Join	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Use Profile Picture
 Select All / Deselect All

Connect your Twitter Account:
Currently Linked Account: **larsfiedler**

Figure 10. Options for linking social media accounts.

3.10.1 Activities

As users create and run applications, or make updates to boards and groups, we record those actions. Users can then view the activities of other users, or other resources. Note that the activities a user can view represent an intersection of the discover permissions they have on the entities and the resources referenced in an activity.

3.10.2 Comments

Users can comment on resources such as applications and boards, and create message threads. Example comments may include: what they are using the application for, what issues they ran into, etc.

3.10.3 Reviews

As the number of applications in the system grows and the similarity of applications blurs, it becomes hard for users to know which application is the ‘best fit.’ Reviews allow the users to rate applications and provide feedback on functionality.

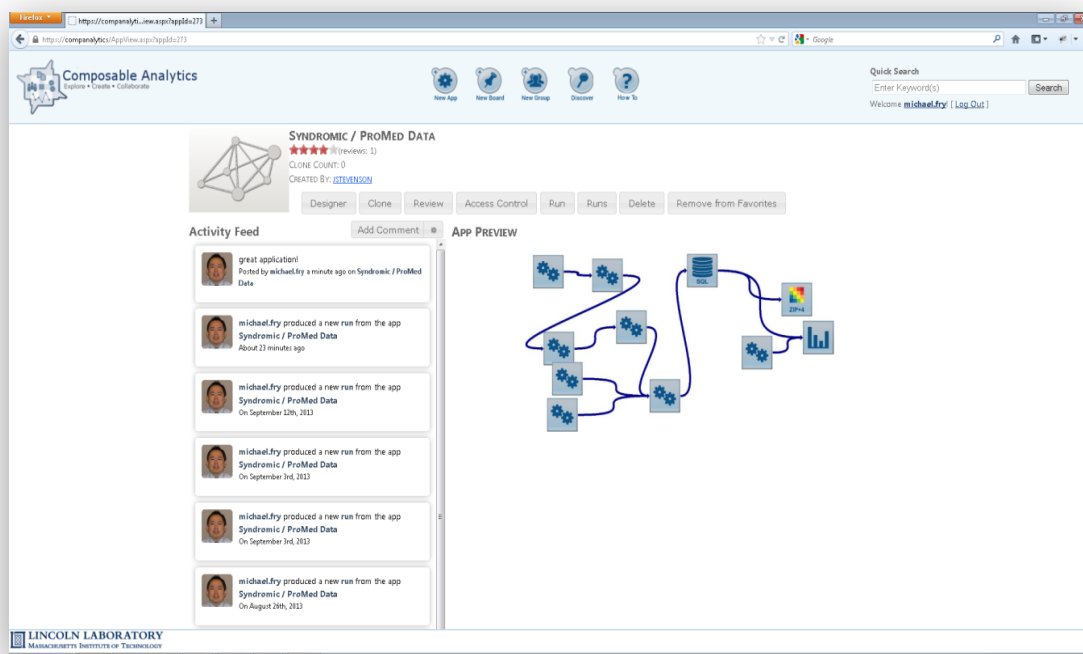


Figure 11. Application landing page shows activities and comments (left), rating (top), and preview (center).

This page intentionally left blank.

4. USABILITY

Usability is an area where we strive for continual innovation. Composable Analytics is nothing without its users. Modifying and creating applications is the most important area for generating new content within Composable Analytics. Users need to easily be able to tweak applications, create new applications, and generate results to share with others. We have created a few methods to help ameliorate the barrier to entry.

4.1 HOW-TO VIDEOS

We have created screen-cast videos demonstrating how to develop applications within Composable Analytics. Because creating an application is very interactive (i.e., drag and drop), a video captures all the nuances that would otherwise be hard to explain through text.

4.2 EXAMPLES APPLICATIONS

When starting to use a module within an application, users may have questions surrounding its functionality. To alleviate some of this initial ambiguity, every module includes an associated example application. With a simple click in the application designer, users can open an example application and understand how the module can be used. All example applications can be executed without any modification.

4.3 DESCRIPTIONS

Each module and its inputs and outputs contain descriptions of their functionality and what values are expected. These can be visible when a user hovers over the module with the mouse. Understanding a module's inputs, its functionality, and what it returns are required before we can expect a user to begin using it successfully in an application.

This page intentionally left blank.

5. SOFTWARE ARCHITECTURE

We are currently using a three-tier architecture: UI/web pages, web services, and a database. We are developing our front-end using .NET Aspx web pages, jQuery, JavaScript, and CSS. The pages are loaded from the web server, which then make Ajax requests to our web services (middle tier). The web services provide a means of serving up data to the web pages and also allows for web clients to save data. The web service layer is also how clients execute applications and get information on the lifecycle of an application. All the web services are hosted in an IIS web server. The processing of triggers occurs in the activation host. The activation host receives emails, web requests and timer events, and runs the corresponding application. The activation host must also communicate with the database to load applications for execution and store the results.

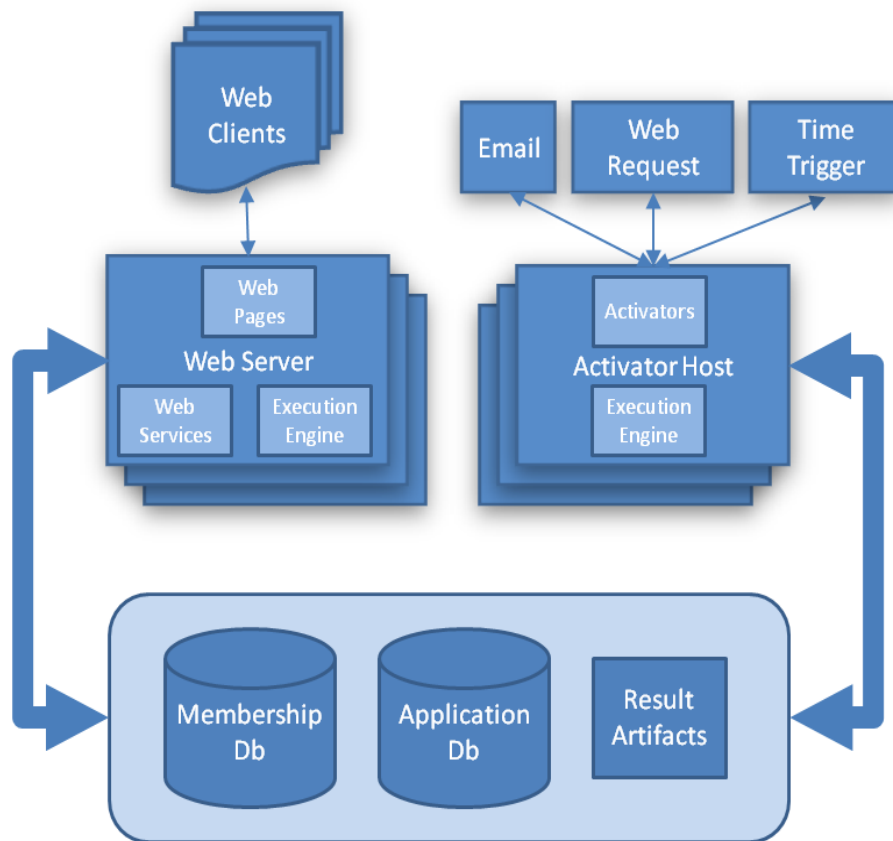


Figure 12. Three-tier architecture: (1) web clients/browsers, (2) web services and activation host, and (3) database.

5.1 PRESENTATION LAYER

The presentation layer consists mostly of JavaScript. We've relied heavily on the jQuery library to support DOM manipulation, third-party plugins, and functional structure.

A script dependency issue initially arose during front-end development. For example, script A may depend on B, which then depends on C. For the page to work, all three would need to be included in the page – resulting in root JavaScript code knowing about dependency C. There are other scenarios where two nondependent pieces have a common dependency, which needs to come first. To alleviate this dependency nightmare and possible double inclusion of scripts, we developed an Asset Service and a dependency chain markup in JavaScript.

If a JavaScript file requires another script to work properly, it places a require comment in its script.

```
//=require otherfile.js
```

In addition, root JavaScript files can be loaded through the Asset Service through seed JavaScript loading functions. The asset service will load these root JavaScript files and look for any require tags. If any of them exist, then those files are loaded before continuing to load the current file. The file will not be reloaded if it has already been loaded (i.e., two files require the same library). This allows scripts to articulate dependencies without each webpage needing to include all the possible dependencies in the correct order.

5.2 WEB SERVICES

All of our web services communicate with the user interface through the JavaScript Object Notation (JSON) wire format. This allows clients and the UI to easily construct requests and parse the results. Windows Communication Framework (WCF) is currently being used for construction of the web services and data contracts.

5.3 DATA LAYER

We are using a SQL Server relational database to store all of our resource data. To interact with the database, we are using the Entity Framework, an object relational mapper (ORM). Note that this data is different from the data that applications typically query, and at some level can be considered system metadata.

Because runs and results are considered securable resources, they are stored in our database. Some results artifacts are stored to disk. Each application run gets a folder where modules can save information. Examples of data saved to this area include: CSV files, images, sql-lite databases.

Below is our entity relational diagram. SQL database tables and classes are automatically generated from this model. We can then use these classes to store data and create queries to retrieve the data.



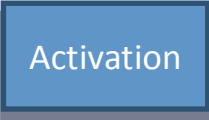

Tier	Container	Technology
 Browser	Firefox, IE, Chrome	HTML, HTML5, JS, CSS, JQuery
 Web Server	Internet Information Systems (IIS)	.NET, WCF, Asp.NET
 Activation	NT Service(s)	.NET
 Databases	Sql Server, Sql-Lite	Entity Framework, Ado.NET

Figure 14. Technologies used.

6. LEN PREPARATION

Composable Analytics will be hosted on the Laboratory's external network (LEN). This will give us an opportunity to get user feedback from the community. The Public Health Community is the first audience we are planning to target.

6.1 FIXING SECURITY VULNERABILITIES

We have currently been in the process of fixing security vulnerabilities in the code base. Some of these changes included protection from SQL injection, locking the system down to only authenticated users, and adjusting the user account creation process.

6.2 CDC MORBIDITY AND MORTALITY WEEKLY REPORT DATA

We have also seeded the LEN environment with CDC's weekly disease counts for each division and state in the United States. While this information is publicly available for anyone to view, little analysis has been done due to the format in which the data is released. This information was placed in a relational database and applications were then created within Composable Analytics to query, analyze, and plot the results.

This page intentionally left blank.

7. FUTURE WORK

7.1 COMMERCIALIZATION

We have started conversations with business and venture related groups on campus, including the Venture Mentoring Services and Innovation Teams. There are wide variety of possible target domains and licensing approaches. Some of the industries that could benefit from Composable Analytics include pharmaceuticals, health care, insurance, actuaries, and financial markets. One great benefit of Composable Analytics is the data and the modules plugged in to the system dictate the applicable domains. We can very easily target multiple domains and quickly create functional demos for potential customers.

While analytics may be a huge market, there is another that may be just as big. This includes automating human workflows, and in particular automating the interaction with web applications. There is a large group of individuals who are looking at social sites and performing multiple queries on a daily basis. With Composable Analytics, these users could create applications to query the services, bring the data together and push out the updates and relevant information. Applications that synchronize social web technologies like Flickr, Facebook, and Twitter could also be developed.

7.2 TECHNICAL FEATURES

7.2.1 Default Permissions

Because our permission model is at the individual resource level, it can be tedious to update permissions for every resource. Users require the ability to update permissions for many entities at once. An example would be, ‘update all the runs for a particular app with group x having read permissions.’

In addition, default permissions currently set the owner of the resource to the author, and no one else is on the permission list. While this is the most secure and alleviates issues where items are unexpectedly shared, it can be tedious to constantly need to change permissions. If a user or system executes an application, the user is required to change the permissions after each run if they want to share the result with someone else. Instead, it would be easier for a user to set default permissions. An example would be, ‘for a specific application, set any future runs to be publicly readable.’

7.2.2 User Resource Quotas

Currently users can create and run as many applications as they want. In addition, there are no disk usage restrictions in place. Most commercial email services place restrictions on inbox sizes. Similarly, we will also need to place limits on the number of applications run, and the amount of disk used for results.

7.2.3 Advisory Services for Applications

7.2.3.1 *Semantic Compatibility*

As the number of modules and applications grows, the opportunities for users to build ‘incorrect’ applications increases. Not all modules can be connected together. Obviously the input and output types have to be well matched; but even modules operating on the same types may not be compatible with each other. Additional data that describes the meaning of the modules may lead to better insight in recognizing application ‘correctness.’

An intuitive approach to evaluating the compatibility of the modules within an application is to, once a reasonable set of module features has been established, employ techniques from machine learning to classify the collection of modules as being a potentially successful or unsuccessful analytic. The module feature space would need to indicate which modules are compatible with analytic contexts, which include statistical analysis, data visualization, signal processing, etc. These features would be discovered through combination of unsupervised learning techniques, such as hierarchical clustering, from module usage statistics as well as indications given by the module developer. Once the classifier is trained on the module feature space, we will be able to, in the event that an application is classified as potentially unsuccessful, issue a warning to the application designer that the combination of modules selected are not likely to produce meaningful analysis. This allows the user to either redesign the application to be more appropriate for the intended task or run the application anyway. We allow this latter function as a means to allow the Composable Analytics user base to help us discover novel uses for existing modules. Provided a sufficient amount of usage statistics, the classification engine deeming an application as successful or unsuccessful can be tuned through observing newer, creative applications in which modules are successfully employed outside of their originally intended analytic context.

Of course, during the alpha phase, there will be an insufficient amount of usage data. Hence, feature space learning and training the classifier used for semantic compatibility evaluation will not be possible. To remedy this dilemma, we will, in early versions of Composable Analytics, use fuzzy logic to evaluate the compatibility of modules within an application. Upon the creation of a new module we will ask the developer to assign the module *membership* to a set of analytic contexts. Using the analytic context memberships of the modules that comprise it, we will determine the membership of the application. In the event that the application has weak membership to any one of the predetermined analytic contexts we will issue the application’s designer a warning. In the event that the designer chooses to run the application in spite of the warning and is pleased with the results, as indicated through the user, say, rating the application highly, we can tune the analytic context memberships of modules that appear in the application accordingly. Once a sufficient amount of usage statistics becomes available, we can either switch to the machine learning approach of compatibility evaluation, or fuse it with the fuzzy logic approach.

7.2.3.2 Application Similarity

Finding applications that have similar structure and behavior can lead to an increase in reuse and give users the ability to find colleagues working on similar solutions. This can also help in discovering applications based on inputs, outputs and behavior.

7.2.3.3 Recommendation Services

There are two notions of recommendation services that we are currently considering. At a basic level, we are investigating popularity metrics for applications and boards. As the number of applications in the system grows, understanding what is popular provides additional measurements in making recommendation decisions. Popularity of an application can be a function of ratings, number of clones, number of runs by unique users, etc. By producing an affine combination of these values for each application or board, we can develop ranked lists of the most popular ones. This allows new users to get a feel for Composable Analytics through pre-vetted entities. Also, popularity rankings provide a foundation on which to build more sophisticated application or board recommendations. For instance, applying filters or keyword searches to the popularity ranking allows users an opportunity to discover existing analyses that may be pertinent to his or her goals.

Additionally, we are developing *helper modules* that future developers can recommend be used in conjunction with the module they are creating. For instance, many statistical analyses and signal processing procedures rely on the assumption that the data under observation is normally distributed. A helper module for one of these modules would be a goodness-of-fit test on the normality of the input data. We intend to provide a pathway for the developer to issue a pop-up message that indicates that the module assumes the input data to be normally distributed and recommend the goodness-of-fit helper module to the application designer.

This page intentionally left blank.

8. FINAL THOUGHTS

We will focus our future attention on three main areas: user outreach, user-experience services, and platform. We need to engage a user base sooner rather than later. Placing Composable Analytics on the LEN will be a key stepping-stone to accomplishing this goal. As our user base grows, we will have more data and insight in developing targeted recommendation and advisory services, and overcoming usability issues. Performance issues will most likely arise as our user-base grows, and the need to redesign and develop certain features will most likely occur. Finally, we will continue to improve the platform and framework so we can build additional features.

This page intentionally left blank.

APPENDIX A
LIST OF MODULES

Name	Description	Category
RabbitMQ Subscriber	Activates the execution of the application by receiving a message on a RabbitMQ topic	activators
Receive Email	Activates an application by receiving an email with the appId in the subject. The subject should be begin with: <appId>	activators
Timer	Activates an application based on the user specified schedule	activators
Web Receive	Activates an application by receiving a web request on the URI ~/services/WebActivationService.svc/Activate?appId=xxx	activators
Web Send	Sets the response to a web request directed to the application	activators
Csv Reader	Reads in delimited data from Uri and returns in table format	datasources
File Uploader	Reads and uploads file from the browser machine to the server, and returns its Uri on the server	datasources
Google Insights For Search	NULL	datasources
Image Fetcher	Retrieves image from specified Uri	datasources
Odbc Insert	Inserts table data into a SQL database table	datasources
Odbc Query	Queries odbc database and returns data as a Table	datasources
Odbc Select Query	Creates a select statement for an odbc database and returns data as a Table	datasources
Reportable Diseases	NULL	datasources

Name	Description	Category
Rss Client	Retrieves Rss or Atom feed from Uri and returns parsed feed information	datasources
Sql Query	Queries sql database and returns data as a Table	datasources
Sql Select Query	Queries sql database and returns data as a Table	datasources
WebClient	Retrieves contents at specified Uri	datasources
XML Reader	Reads xml from string and returns in tabular format at specified root	datasources
Age Groups	Selects CDC Age Groups	health
CDC MMWR Disease	NULL	health
CDC MMWR Region	NULL	health
Date to Week and Year	Converts DateTime to an Epi Week number and Year	health
DiseaseToSyn	Converts a Disease to a list of typical syndromes for the disease	health
Florida Counties	NULL	health
HealthMap	NULL	health
Syndromes	NULL	health
Syndromic Data	NULL	health
Syndromic Data What	NULL	health
Syndromic Data Where	NULL	health
Highchart Scatter Series Input	NULL	inputs
Highchart Series Input	NULL	inputs
String Input	Simply forwards input string to result, allowing for an	inputs

Name	Description	Category
	input to be specified in separate module	
Image Hist Equalizer	NULL	MATLAB
Image Intensity	NULL	MATLAB
MATLAB	Executes MATLAB code	MATLAB
MATLAB Get Variable	Returns a variable from the MATLAB workspace	MATLAB
MATLAB Put Variable	Sets a variable in the MATLAB workspace	MATLAB
Plotter	Constructs a plot using MATLAB	
Array Indexer	Returns object at specified index	operators
Branch	Conditionally executes modules connected to the then and else results	operators
Calculator	Performs numerical operation on two numbers	operators
Code	Executes a user coded function in a sandbox	operators
Consolidator	Takes in a list of optional inputs and will return the first one in the list. Not all input connections need to return a value.	operators
Date/Time Calculator	Performs timespan operation on a date/time	operators
Key Value Pair	Constructs a KVP object from incoming Key and Value	operators
Replacement Rule	Creates a dictionary of two strings (value and reference value) from two lists of strings	operators
Sql Conditional	Combines multiple sql clauses using the specified condition	operators
Sql Expression	Constructs a sql clause using column names, operators and value	operators

Name	Description	Category
Sql Like Operator	Produces SQL clause by concatenating column name and values with the 'like' operator	operators
Sql Operator	Constructs a sql clause using name, operator and value	operators
SQL Replacement	Generates SQL query for replacing column values	operators
Sql Timestamp Expression	Constructs a sql clause using column names, operators and value	operators
State To Counties	Returns counties in a state	operators
String Formatter	Combines multiple strings together using the specified format	operators
SyndicationToTable	Create a table structure from syndication feed	operators
Syndromic Data Demo	Queries syndromic data by disease, county, and date	operators
Uri Builder	Creates a Uri object	operators
Uri Param Parser	Creates a Uri object	operators
Board Publisher	Sends the incoming result to the board	operators
Csv Writer	Writes a table to a file in the Comma Separated Value format	outputs
Facebook	Publishes a status update to Facebook	outputs
Mail Sender	Sends an email to specified addresses	outputs
RabbitMQ Publisher	Publishes a message onto a RabbitMQ topic	outputs
Twitter	Publishes a status update to Twitter	outputs
Lilliefors Normality Test	Uses Lilliefors hypothesis test to determine if a data set is normally distributed	statistics
TwoByTwoTable	NULL	statistics

Name	Description	Category
Column Type Converter	Converts table columns to the desired type.	tables
Table Aggregator	Aggregates the data based on columns and operation	tables
Table Column Reducer	Returns a table containing only the columns specified from the original table.	tables
Table Column Type Input	Returns user specified table column type	tables
Table Filter	Filters the data based on clauses	tables
Table Filter Logic Clause	Filters the data based on multiple clauses	tables
Table Filter Operator Clause	Filters the data based on operation	tables
Table Scaler	Scales a column of a table by a column in another table	tables
Table Set Operation	Performs a set operation on a collection of tables	tables
Table Sort Column Input	Returns user specified sorting direction of a table column	tables
Table Sorter	Sorts a table by column	tables
TableColumn	Pulls out a particular column from a table	tables
TableCreator	Creates table based on inputs	tables
CountyHeatMap	Creates Kml Heat Map from county tabular data	visualizers
DivisionHeatMap	Creates Kml Heat Map from division tabular data	visualizers
Google Charts	NULL	visualizers
Highchart Bar Chart	NULL	visualizers
Highchart Line Chart	NULL	visualizers

Name	Description	Category
Highchart Scatter Chart	NULL	visualizers
Highchart to Image	NULL	visualizers
Kmz	Zips up multiple Kml files into a single Kmz file	visualizers
StateHeatMap	Creates Kml Heat Map from state tabular data	visualizers
Table to Kml	Creates a Kml file of placemarkers using data in the specified table	visualizers
ZipCodeHeatMap	NULL	visualizers

REPORT DOCUMENTATION PAGE*Form Approved*
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 29 April 2014		2. REPORT TYPE Project Report		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE Systems and Methods for Composable Analytics				5a. CONTRACT NUMBER FA8721-05-C-0002	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
Lars H. Fiedler and Timothy J. Dasey				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
MIT Lincoln Laboratory 244 Wood Street Lexington, MA 02420-9108				CA-1	
Assistant Secretary of Defense, Research and Technology 4800 Mark Center Drive Suite 16F09-02 Alexandria, VA 22350-3600				10. SPONSOR/MONITOR'S ACRONYM(S) ASD(R&E)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
Approved for public release; distribution is unlimited.					
Composable Analytics is a web-based software platform that enables community researchers, analysts, and decision makers to collaboratively explore complex, information-based problems through the creation and use of customized analysis applications.					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as report	18. NUMBER OF PAGES 42	19a. NAME OF RESPONSIBLE PERSON
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (include area code)

This page intentionally left blank.

