5-2022

Proposed Application for an Entity Component System in an Energy Services Interface

Tylor Slay
Portland State University, tylor.slay@gmail.com

Grace B. Spitzer
Portland State University

Robert B. Bass
Portland State University, robert.bass@pdx.edu

Proposed Application for an Entity Component System in an Energy Services Interface

Tylor Slay, Grace B. Spitzer, Robert B. Bass
Department of Electrical & Computer Engineering
Portland State University
Portland, OR, USA
tslay@pdx.edu

***Abstract*—An Entity Component System is a data-oriented architecture originally developed to streamline video game performance. Despite being quite new, Entity Component Systems are relatively well established within the video game industry due to the cutting edge nature of research into performance, especially around graphics. However, Entity Component Systems have not been widely examined or adopted outside of that industry. We propose adopting an Entity Component Systems framework to serve the needs of an Energy Service Interfaces. We examine the needs of an Energy Service Interface, give an overview of open-source Entity Component Systems (ECSs) libraries, examine some preliminary performance results for ECSs, and explore the traditional approach to fulfilling the needs of an Energy Service Interface (ESI) with database architectures.**

***Index Terms*—Energy Service Interface, Distributed Energy Resource, Smart Energy Profile, Data-Oriented Design, Entity Component Systems**

I. Introduction

The mass adoption of renewable energy generating resources has lead to the need for increased flexibility of the bulk power system. Traditionally, deterministic generation has followed stochastic load. The new paradigm in the bulk power system will be deterministic load following stochastic generation, in addition to a reduced fraction of traditional generation following load. This will require new technologies to promote participation from a wide range of devices.

The ESI, first presented by Harden in 2011, is an enabling contribution towards this new paradigm [1]. The objective of the ESI is to ensure secure, trustworthy information exchange between utilities and customer-owned Distributed Energy Resources (DERs) in order to promote dispatch of grid services through large-scale deployment of DERs. The ESI does so by providing a set of rules and interoperability requirements that define bi-directional, service-oriented, logical interfaces with expectations for privacy, security, and trust. The Grid Modernization Laboratory Consortium (GMLC) has built upon Harden's ESI concept and has demonstrated a vision of the future where major household appliances are grid-enabled DERs that contribute to the reliability and resiliency of the bulk power system [2].

An ESI server would necessarily require some method of storing data relating to the aggregation of DERs. Traditionally, a database would serve this role. The

This work was supported by US DOE OE0000922.

server performance would thus be limited by the speed of database access [3].

A high performance back-end database is necessary for large-scale DER aggregation. Traditional databases are well suited for static storage of data at the scale expected of an ESI. However, the memory demands of an ESI server are not static. In response to the need to continuously modify data, a traditional database has severe limitations, such as query times and data translation for processing. ECSs are a data-oriented design paradigm that could potentially improve upon database performance in these areas.

Entity Component System were developed to streamline memory management in video games. In a field where cutting edge graphics are a huge competitive advantage, video game developers have begun adopting ECSs for data management. ECSs continuously process enormous numbers of individual entities whose characteristics are both dynamic and interrelated. The approach taken by ECS offers significant benefits in terms of individual entity query times and data processing tasks when compared to databases. The use of ECSs is well established in the video game industry, with major corporations such as Unity[1] having adopted ECS frameworks for their software products. Outside the video game and VFX industries, however, ECS have seen little to no attention or adoption. This is partially because of the highly specialized nature of the technology and development thus far, and partially because adoption in industries outside of software moves at a more conservative pace. Given this context, we propose that the use case for this new technology could very well be more in line with the memory management needs of an ESI than a traditional database.

II. ESI Service Requirements

Electric utilities are responsible for maintaining power system operations within physical constraints to prevent damage to system component, and to ensure stable, reliable and economical delivery of electricity to customers. Grid services are a means by which a utilities achieves these operational objectives. DERs may be used to provide grid services through coordinated aggregation and dispatch using a DER Management System (DERMS). In this work, the ESI server is the DERMS [4].

There are myriad energy services used by utilities [5]. Some services balance energy generation and demand on a day-ahead basis while others are autonomous controls that balance on a sub-second basis. For this paper we use as a working example the CAISO EIM, which uses five minute control signals to balance generation and demand.

Figure 1 shows an approximated EIM signal using the difference between the hour-ahead forecast and real-time demand.[2] This representation of an EIM signal is sufficient for approximating DER participation. It should be noted the approximated EIM follows a *generation-following-load* control paradigm, where the grid operator needs to ensure enough generation is available to meet load. By assuming a *load-following-generation* paradigm, the ESI server will increase demand to meet generation by dispatching DER loads. This is the paradigm we have chosen to implement for our approximated EIM signal.

Figure 2 demonstrates the required number of DERs to meet the service commitment of our approximated EIM signal. The schedule updates indicate how many schedules will need to be updated between each 5 minute increment. A negative schedule update represents

[1]https://docs.unity3d.com/Packages/com.unity.entities@0.17/manual/index.html

[2]http://www.caiso.com/TodaysOutlook

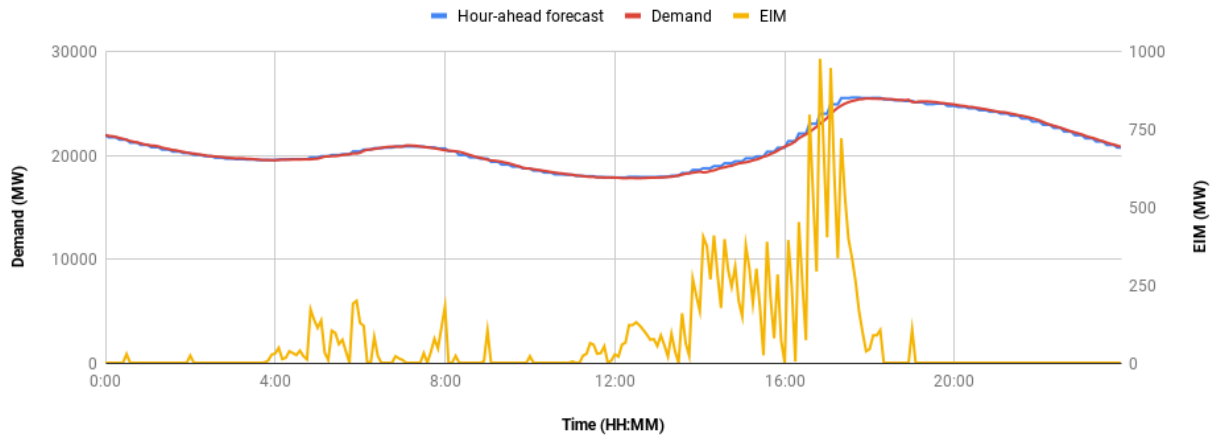

Fig. 1. CAISO hour-ahead demand forecast (blue) and real-time demand (red), left axis, for Jan 01, 2021 in 5 minute increments. Deviations between hour-ahead and demand occur throughout the day, and result in EIM resource dispatch (yellow), right axis, if demand falls below forecasts. The EIM dispatch is set to zero when demand is greater than forecasts because residential DER loads do not produce power.

the number of active schedules that will need to be rescheduled to a later time. The DERs used for this approximation were all assumed to be electric water heaters with a 4500 watt power draw [6]. It is reasonable to assume there will always be enough DERs to meet the approximated EIM service commitment based on the projected adoption of smart water heaters [7], [8].

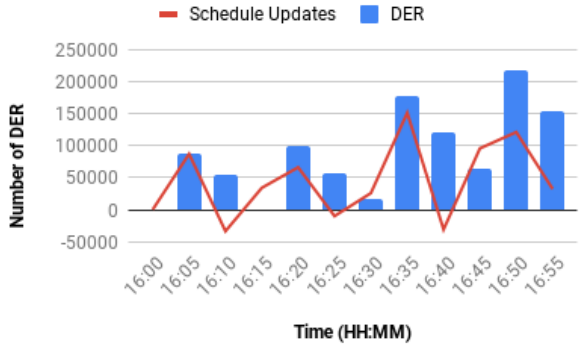

Fig. 2. Number of DERs required to achieve EIM service commitments. Negative schedule updates indicate the number of active schedules that need to be rescheduled to a later time.

The average number of scheduled updates for the approximated EIM service is nearly 100,000. This number does not reflect the process of checking currently scheduled services and updating the amount of energy available for participation given how long that specific DER has been active. It also doesn't reflect how many total DERs will be participating in other services for the grid. An ESI server will need to continually process incoming DER participation requests, optimize scheduling for service, and estimate available energy for participation. In the next section, we will discuss current back-end database solutions for managing data.

III. Database Back-end

Websites today are rarely hard-coded HTML. Most servers use some back-end database that manages requests for information to be displayed on a page. This back-end database typically consists of a lower-level programming language that queries structured data stored in a database. Most databases fall into the relational (SQL) and non-relation (NoSQL) types.

The primary differences between relational and non-relation databases are the way data are organized and how the databases scale. Čerešňák and Kvat note that

SQL databases use a predefined scheme for data organization and scale vertically with more memory and processing power [9]. NoSQL databases use a dynamic scheme for data organization and scale horizontally with more servers rather than increasing the processing and memory capabilities of a single server. Databases are a well established technology, with a large body of peer-review literature devoted to examining their performance in depth. Table I shows performance comparison of several common relational and non-relational databases [9].

Database	Insert	Update	Delete	Select
Oracle	0.091	0.092	0.119	0.062
MySQL	0.038	0.068	0.047	0.067
MsSQL	0.093	0.075	0.171	0.060
Mongo	0.005	0.009	0.015	0.009
Redis	0.010	0.013	0.021	0.015
GraphQL	0.008	0.012	0.018	0.011
Cassandra	0.011	0.014	0.019	0.014

TABLE I
QUERY PERFORMANCE (MILLISECONDS) OF DATABASE WITH 100,000 RECORDS. [9]

The approximated ESI server requirements outlined in the previous section show a typical database would be adequate for scheduling DERs for EIM service participation. However, as the number of participating DERs increases, the performance of the back-end database will become a significant bottle-neck within the ESI server. Kepner et. al. show that reaching database updates into the millions per second is not a trivial feat [10]. While technology has advanced in the recent years, even Google has only been able to achieve one million inserts per second in 2014. In the following section, we explore the new data-oriented paradigm in programming and how new frameworks in video games are allowing much faster processing of large amounts of data.

IV. ENTITY COMPONENT SYSTEM

With Object-Oriented Programming (OOP), an entity has an inherited type, and there are often multiple levels of inheritance. Managing the complexities associated with these inheritance types increases the overhead of OOP, both in terms of development and run-time performance.

Unlike OOP, ECSs are based on composition rather then inheritance. Individual entities within a system are composed of various pieces of information, known as components. The idea behind data-oriented design, the umbrella term under which Entity Component Systems fall, is to bypass the added complexity of inheritance. By arranging all data into entities with associated components, the designer has the ability to streamline multiple types of operations within the system as a whole. Added to this, it is possible to implement an ECS so that sets of entities that share components are stored contiguously in memory, additionally increasing the performance of the system when those components need to be accessed or modified.

Figure 3 outlines how an ESI client would implement the ECS. For this example we use *flow reservation request* and *flow reservation response* as the primary entities used for communication between clients and the ESI server. Each of these entities is comprised of several components including: interval, power, energy, and their respective polling rate to name a few. The example implementation has three systems to manage these entities: polling, request, and response. These systems are responsible for ensuring the entities are up-to-date and if they are not, then the entities are updated to ensure active participation in the ESI server services.

ECSs have been used in video games to increase performance by modeling objects within a game as entities. Trees, cars, nonplayer characters (NPCs), and

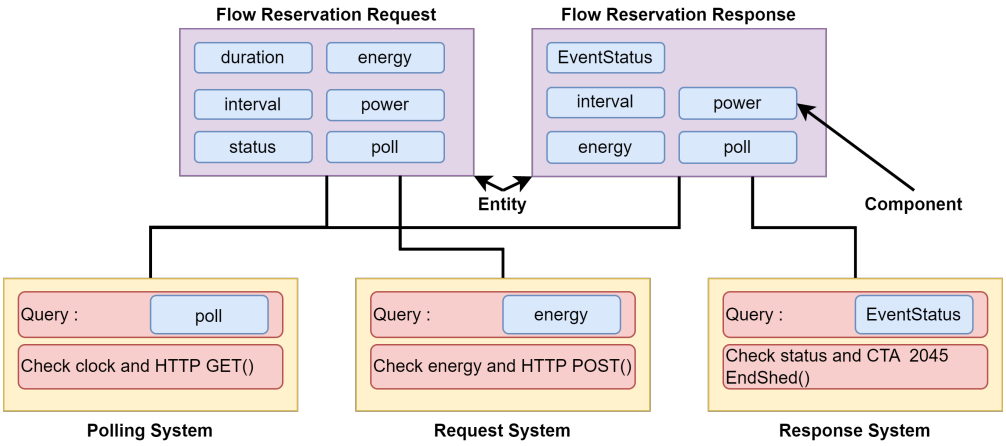

Fig. 3. Example implementation of an Entity Component System within the ESI server/client system. *Entities* (purple) represent resources exchanged between the ESI server and DER clients, each of which contain unique sets of data known as *components* (blue). Different *systems* (orange) call entities to query component values or execute functions (red). In an ECS, data are decoupled from functions, in contrast to an OOP where these exist together within classes.

bullets are all represented as entities, each of which must be rendered within the visualization space. When rendering a frame in this space, operations are performed on sets of entities, such as all entities that should move or collide. Each successive frame must be re-rendered, which places considerable demand on data management. This is very similar to the needs of an ESI server, which might, for example, have to update the schedules of hundreds of thousands of DERs that are being dispatched to provide a grid service.

Table II shows a performance comparison between several open-source ECS libraries [11]: EnTT, EntityX and ECS-lib. These results are from a Safe Access test. For this experiment, each entity has a position, velocity, and target reference. An "access," of an entity in this case means that the entity was given a randomly generated target. The entity first checks to see if the target is "alive," (thus Safe rather then Unsafe Access). The entity then alters its velocity to point towards the target, assuming the target is "alive."

Of the open-source libraries tested, EnTT and EntityX are more traditional data-oriented implementations [11].

Library	**1,000**	**10,000**	**100,000**	**1,000,000**
EnTT	0.022958	0.024666	0.039306	0.0913204
EntityX	0.08049	0.0864236	0.111046	0.164747
ECS-lib	0.42998	5.78322	58.8688	N/A

TABLE II

AVERAGE TIME/ENTITY (MICROSECONDS) FOR SAFE ACCESS TEST FOR INCREASING DATASET SIZES (ENTITY COUNT).

They both use integers to represent entities, and store a version with each entity. They have equivalent O(1) performance to check if an entity still exists, meaning it is "alive." EnTT uses a continuous array for each type of component, where the integer representation of the entity to which that component belongs is used as a key to find the index of the correct component. EntityX also stores components in continuous arrays, but the arrays have space for every entity to contain every component, regardless of whether they actually do or not. The ECS-lib library, contrary to its name, varies from the other two libraries in that its architecture is more heavily influenced by object-oriented design then data-oriented design. Each entity is itself responsible for managing its

own data, which is distinctly antithetical to data-oriented principles, and hence its much slower performance in all scenarios.

Note that the hardware used for the experiment whose values are shown in Table I was not equivalent to the hardware used for the experiment that produced the performance numbers in Table II, and thus a direct side-by-side comparison of the values would not be a fair assessment of performance [9], [11]. The performances metrics of the ECS frameworks still demonstrate a processing time that is several orders of magnitude faster than the typical back-end database solution.

V. Conclusion

This paper proposes the use of an ECS to implement a high performance ESI server. Limited existing research suggests that ECSs could offer significant performance gains compared to databases, especially when taking into consideration the specific requirements of an ESI server.

The predicted benefits of an ECS architecture are based on reasonable assumptions, but have thus far been overlooked when it comes to more rigorous experimental verification. In the literature search for this paper, very little work regarding ECS performance analysis was found. One possible reason for this might be the lack of focus on peer reviewed publication on the part of video game developers. For whatever reason, no close comparison of performance seems to currently exist between databases and ECSs. Therefore, the numbers given in this work come from separate experiments that were not designed to accurately measure performance differences between databases and ECSs. Nonetheless, the authors believe that they effectively demonstrate the potential performance advantages of ECSs for an ESI. The authors believe that further work, including a direct experimental comparison of database and ECS performance, is warranted.

References

[1] D. Hardin, "Customer energy services interface white paper," in *Gridinterop Forum*, 2011.

[2] S. Widergren, R. Melton, A. Khandekar, B. Nordman, and M. Knight, "The plug-and-play electricity era: Interoperability to integrate anything, anywhere, anytime," *IEEE Power & Energy Mag.*, vol. 17, no. 5, pp. 47–58, 2019.

[3] S. Malkowski, M. Hedwig, and C. Pu, "Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks," in *IEEE Int. Symp. on Workload Characterization*, 2009, pp. 118–127.

[4] T. Slay and R. Bass, "An energy service interface for distributed energy resources," *IEEE Conf. on Tech. for Sustainability*, 2021.

[5] M. Obi, T. Slay, and R. Bass, "Distributed energy resource aggregation using customer-owned equipment: A review of literature and standards," *Energy Reports*, vol. 6, pp. 2358–2369, 2020.

[6] T. Clarke, T. Slay, C. Eustis, and R. B. Bass, "Aggregation of residential water heaters for peak shifting and frequency response services," *IEEE Open Access Journal of Power and Energy*, vol. 7, pp. 22–30, 2019.

[7] Boneville Power Administration, "CTA-2045 Water Heater Demonstration Report: A Business Case for CTA-2045 Market Transformation," November 2018.

[8] K. Marnell, C. Eustis, and R. Bass, "Resource study of large-scale electric water heater aggregation," *IEEE Open Access J. of Power & Energy*, vol. 7, pp. 82–90, 2020.

[9] R. Čerešňák and M. Kvet, "Comparison of query performance in relational a non-relation databases," *Transportation Research Procedia*, vol. 40, pp. 170–177, 2019.

[10] J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, P. Michaleas, J. Mullen, A. Prout, A. Reuther, A. Rosa, and C. Yee, "Achieving 100,000,000 database inserts per second using Accumulo and D4M," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, 2014, pp. 1–6.

[11] H. Hansen and O. Öhrström, *Benchmarking and Analysis of Entity Referencing Within Open-Source Entity Component Systems*. Malmö Universitet, 2020.