A Framework for Estimating the Energy Consumption Induced by a Distributed System's Architectural Style

Chiyoung Seo[1], George Edwards[2], Daniel Popescu[2], Sam Malek[3], Nenad Medvidovic[2]

[1]Yahoo! Inc.
701 First Ave, Sunnyvale, CA 94089, U.S.A.
chiyoungseo@yahoo.com

[2]Computer Science Department, University of Southern California
Los Angeles, CA 90089-0781, U.S.A.
{gedwards,dpopescu,neno}@usc.edu

[3]Department of Computer Science, George Mason University
Fairfax, VA 22030-4444, U.S.A.
smalek@gmu.edu

Abstract. The selection of an architectural style for a given software system is an important factor in satisfying its quality requirements. In battery-powered environments, such as mobile and pervasive systems, efficiency with respect to energy consumption has gained prominence as an important quality requirement. In this paper, we present a framework that facilitates early estimation of the energy consumption induced by an architectural style in a distributed system, and enables an engineer to use energy consumption estimates along with other quality attributes in determining the most appropriate style for a given distributed application. We apply the framework to three architectural styles, and evaluate it for precision and accuracy using a middleware platform that supports the implementation of those styles. In a large number of application scenarios, our framework exhibited excellent precision, in that it was consistently able to correctly rank the styles and estimate the relative differences in their energy costs. Moreover, the framework has also proven to be accurate: its estimates were within 7% of each style implementation's actual energy cost.

1. INTRODUCTION

A promising approach to addressing the challenges of developing distributed, mobile, and pervasive systems is to employ the principles of software architecture [9,12]. Software architecture provides abstractions for representing the structure, behavior, and key properties of a software system [20]. These abstractions include *software components* (computational elements), *connectors* (interaction elements), and *configurations* (specific assemblies of components and connectors). *Architectural styles* (e.g., publish-subscribe, peer-to-peer, client-server) are key design idioms which further refine the vocabulary of components and connectors and propose constraints on how they may be integrated.

Architectural decisions made early in the design process are critical to the successful development of a distributed system. In particular, selecting an appropriate architectural style has a significant impact on system quality attributes (e.g., latency, scalability, reliability, etc.). *Energy efficiency* is increasingly being defined as an important quality attribute for mobile and pervasive applications. However, there are currently no techniques for analyzing the impact of an architectural style on a system's energy consumption. In fact, unlike other quality attributes, such as those mentioned above, a style's energy consumption characteristics are not understood even in an informal and intuitive manner. In this paper, we address this problem via a framework that estimates the impact of a system's architectural style on the system's energy consumption. The framework is intended to be used during architectural design and enables an engineer to use energy consumption estimates, along with other quality attributes, in determining the most appropriate style for an application.

Figure 1 depicts the process and key artifacts employed by the framework. The framework defines a method to derive platform- and application-independent equations that characterize a style's energy consumption behavior. We refer to the equations for a given style as that style's *energy cost model*. Comparing the models of different styles yields insights into the essential differences between the energy costs induced by each style. We have derived the energy cost models for five architectural styles: client-server, peer-to-peer, C2, publish-subscribe (pub-sub), and pipe-and-filter[18]. For brevity, this paper shows the derivation of only the client-server and pub-sub styles.

The framework also defines a process for applying any style's energy cost model to a given distributed system design, prior to system implementation. As we demonstrate, this is accomplished by gathering basic information about the target platform and the system design, and plugging these parameters into the energy cost model. We refer to an energy cost model that has been parameterized in this way as a *energy prediction model*. Energy prediction models enable scenario-specific comparisons of the energy costs of different styles, enabling architects to weigh trade-offs and determine the circumstances under which one style will be more efficient than another. We have created energy prediction models for four different distributed systems. For each system, we chose a set of candidate styles that were appropriate for the system, derived the energy prediction

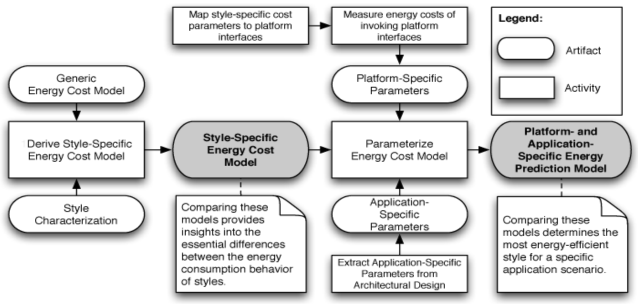

Map style-specific cost parameters to platform interfaces
Measure energy costs of invoking platform interfaces
Legend:
Artifact
Activity
Generic Energy Cost Model
Platform-Specific Parameters
Derive Style-Specific Energy Cost Model
Style-Specific Energy Cost Model
Parameterize Energy Cost Model
Platform- and Application-Specific Energy Prediction Model
Style Characterization
Comparing these models provides insights into the essential differences between the energy consumption behavior of styles.
Application-Specific Parameters
Extract Application-Specific Parameters from Architectural Design
Comparing these models determines the most energy-efficient style for a specific application scenario.

Figure 1. High-level view of the energy consumption estimation framework.

model for each candidate style, and evaluated the framework's accuracy and precision under multiple execution scenarios.

In a recent short paper [17], we outlined the overview of our approach. This paper provides a detailed explanation of the framework and presents the evaluation results. In a large number of experiments, our framework exhibited excellent precision, in that it was correctly able to determine the scenarios under which one style is more efficient than another, and estimate the relative differences in their energy costs. The framework also proved to be accurate: it consistently produced energy consumption estimates that differed from the actual measured energy costs by at most 7%.

This paper is organized as follows: Section 2 describes how our framework defines energy cost models for architectural styles and derives the energy cost models for the client-server and pub-sub styles. Section 3 discusses how energy cost models can be applied to a distributed system, and applies the pub-sub energy cost model to an example application. We present evaluation results in Section 4. Section 5 describes related research, and Section 6 concludes the paper.

2. ENERGY COST MODELS

Fielding [2] and Mehta [13] identified more than twenty architectural styles for distributed systems. In this section, we first show a uniform way to derive a style's *energy cost model*, which is a symbolic expression that represents the energy cost induced by using the style. Then, we illustrate the approach by deriving the energy cost models for the client-server and pub-sub styles, which embody a diverse set of distributed systems characteristics, such as distribution, concurrency, and so on. The derivation of other styles is given in [18].

2.1 Generic Energy Cost Model

As it is common in power modeling of operating systems [10, 23], our energy cost model consists of multiple linear equations. We model the energy consumed by a distributed system as the sum of the energy consumed by its constituent *n* components and *m* connectors, as shown in Equation 1.

$$overallEC = \left(\sum_{i=1}^{n} EC(Comp_i)\right) + \left(\sum_{j=1}^{m} EC(Conn_j)\right) \qquad \textbf{\textit{Eq. 1}}$$

The energy cost of a component $Comp_i$ can be expressed as shown in Equation 2. In this equation, $E_{logic,i}$ is the computational energy cost of the component $Comp_i$ due to executing its core business logic, while $E_{commWithConn,i}$ represents the energy cost of exchanging data via connectors.

$$EC(Comp_i) = E_{logic,i} + E_{commWithConn,i} \quad \textbf{\textit{Eq. 2}}$$

In this work, we assume that a component's core business logic remains the same for all styles. We acknowledge that this logic may need to be refactored in some cases. For example, the logic required by a component to manage its interfaces might differ among styles. We account for these differences in $E_{commWithConn,i}$ of Equation 2. This implies that the computational energy cost of a component (i.e., $E_{logic,i}$ in Equation 2) remains the same across all candidate styles, so our framework does not require the actual value of $E_{logic,i}$ to compare the energy consumption of multiple styles.

Similarly, the energy consumption of a connector $Conn_j$ can be expressed as in Equation 3.

$$EC(Conn_j) = E_{comm,j} + E_{logic,j} \quad \textbf{\textit{Eq. 3}}$$

$E_{comm,j}$ represents the energy consumption of communication, which includes the cost of exchanging data both locally or remotely. We can calculate $E_{comm,j}$ as shown in Equation 4. $E_{commWithComp,j}$ represents the energy consumed by exchanging data with components, while $E_{remoteComm,j}$ and $E_{localComm,j}$ are the energy costs of exchanging data with remote and local connectors, respectively. The above formulation assumes that components and connectors run as separate processes and the energy consumed by one component or connector is not dependent on the energy consumed by other components and connectors. As a consequence, our energy cost model is most accurate for systems where computing resources (such as processor time and memory) are abundant and are assigned "fairly" among all processes. Equation 4 also assumes that component-connector interactions are supported by an Inter-Process Communication (IPC) mechanism, which incurs energy overhead in both the component and connector [23]. If that is not the case, then the value of either $E_{commWithConn,i}$ from Equation 2 *or* $E_{commWithComp,j}$ from Equation 4 would be zero.

$$E_{comm,j} = E_{commWithComp,j} + E_{remoteComm,j} + E_{localComm,j} \quad \textbf{\textit{Eq. 4}}$$

Equation 5 shows how to determine $E_{logic,j}$, which is the energy cost of services (other than communication) that a connector may provide [14]:

- *Coordination* – A connector may transfer execution control among components.
- *Conversion* – A connector may adapt the interface and data provided by one component to that required by another.
- *Facilitation* – A connector may mediate and streamline component interaction.

$$E_{logic,j} = E_{coord,j} + E_{conv,j} + E_{facil,j} \quad \textbf{\textit{Eq. 5}}$$

By combining the above equations, we arrive at the generic energy cost model given in Equation 6. The parameters in Equation 6 vary according to style. In the following subsections, we illustrate how we can create style-specific energy cost models for the client-server and pub-sub styles by refining these parameters.

$$overallEC = \left(\sum_{i=1}^{n} E_{logic,i} + E_{commWithConn,i}\right) + \left(\sum_{j=1}^{m} \begin{matrix} E_{commWithComp,j} + E_{remoteComm,j} + \\ E_{localComm,j} + E_{coord,j} + E_{conv,j} + \\ E_{facil,j} \end{matrix}\right) \quad \textbf{\textit{Eq. 6}}$$

2.2 Client-Server Energy Cost Model

To refine the energy cost parameters introduced in our generic energy cost model shown in Equation 6, we characterize the communication, coordination, conversion, and facilitation required by the client-server style. Figure 2 shows an example of a distributed system designed in the client-server style. Connectors in this style are commonly implemented as middleware stubs and skeletons. The client-server style behaves as follows:

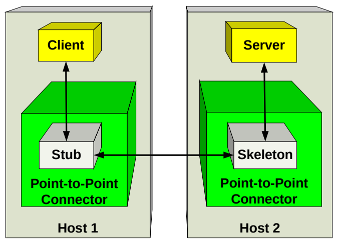


Figure 2. A distributed client-

Component Communication: Clients send requests to and receive responses from client connectors. Servers exchange requests and responses with server connectors.

Connector Communication: Client connectors receive requests from clients and forward them to the appropriate server connector, and receive responses from server connectors and return them to clients. Server connectors receive and buffer requests from client connectors and forward them to servers, and receive responses from servers and return them to client connectors.

Connector Coordination: Connectors transfer execution control by passing requests and responses to clients and servers.

Connector Conversion: Connectors marshal/unmarshal requests and responses.

Connector Facilitation: Connectors create and manage connection objects that implement remote communication.

Equation 7 shows the energy cost $E_{commWithConn,i}$ of a client $Comp_i$ due to sending requests to and receiving responses from a connector. a_i is the total number of requests made by the client. $E_{toConn,k}$ and $E_{fromConn,k}$ represent the energy costs due to sending the kth request to and receiving its response from the connector, respectively. $E_{commWithConn,i}$ of a server $Comp_i$ can be calculated in the same manner using Equation 7. The values of E_{toConn} and $E_{fromConn}$ depend on the platform-specific communication mechanism used between the relevant component and connector. For example, if a queue is used, the values of E_{toConn} and $E_{fromConn}$ are constant; on the other hand, if IPC is used, they are directly proportional to the size of exchanged data [23]. We show how these values can be determined for a given platform in Section 3.

$$E_{commWithConn,i} = \sum_{k=1}^{a_i} \left(E_{toConn,k} + E_{fromConn,k} \right) \quad \textbf{\textit{Eq. 7}}$$

The energy cost $E_{commWithComp,j}$ of a client connector $Conn_j$ incurred by receiving requests from and forwarding responses to clients can be calculated using Equation 8. b_j is the total number of requests received from clients, while $E_{fromComp,l}$ and $E_{toComp,l}$ represent the energy costs due to receiving the lth request and sending the lth response. We can also calculate $E_{commWithComp,j}$ for a server connector $Conn_j$ using Equation 8. Again, the values of $E_{fromComp}$ and E_{toComp} depend on the platform-specific communication mechanism used.

$$E_{commWithComp,j} = \sum_{l=1}^{b_j} \left(E_{fromComp,l} + E_{toComp,l} \right) \quad \textbf{\textit{Eq. 8}}$$

Data exchange between client and server connectors may be either local (if they reside on the same host) or remote, and we handle each case differently. Due to space constraints, we only show how to model the remote communication here. The local case is given in [18]. To model the energy consumption due to *remote* exchange of data between client and server connectors, we assume that the total energy utilization is proportional to the size of the exchanged data. This has been shown to be a highly accurate characterization for commonly used network protocols, such as UDP [1]. Based on this, $E_{remoteComm,j}$ of a client connector $Conn_j$ due to sending c_j requests and receiving responses can be estimated as shown in Equation 9. $tSize_l$ and $rSize_l$ are the sizes (e.g., *KB*) of the lth transmitted request and its received response. *tEC* and *rEC* are the energy costs (*Joule*/*byte*) on the connector's host while it transmits and receives a unit of data, respectively. *tS* and *rS* represent constant energy overheads associated with channel acquisition [1]. All of these values can be easily measured for a given implementation platform, as we demonstrate in Section 3.

$$E_{remoteComm,j} = \sum_{l=1}^{c_j} \left(\left(tSize_l \times tEC + tS \right) + \left(rSize_l \times rEC + rS \right) \right) \textbf{\textit{Eq. 9}}$$

Similarly, we can calculate $E_{remoteComm,j}$ of a server connector $Conn_j$ using Equation 10. d_j is the total number of requests received over the network, and $rSize_l$ and $tSize_l$ are the sizes of the lth received request and its transmitted responses, respectively. $E_{buffer,l}$ is the energy cost of buffering the lth received request.

$$E_{remoteComm,j} = \sum_{l=1}^{d_j} \left(\begin{array}{l} \left(tSize_l \times tEC + tS \right) \\ + \left(rSize_l \times rEC + rS \right) + E_{buffer,l} \end{array} \right) \quad \textbf{\textit{Eq. 10}}$$

In the client-server style, the energy cost $E_{coordin,j}$ due to performing coordination is not modeled separately because connectors transfer execution control by passing requests and responses, whose energy cost is already captured by the $E_{commWithComp,j}$.

The conversion cost $E_{conver,j}$ of a client connector can be quantified as in Equation 11. c_j is the number of requests sent remotely, while $E_{mar,l}$ and $E_{unmar,l}$ are the energy costs of marshalling the lth request and unmarshalling its response, respectively. The conversion cost $E_{conv,j}$ of a server connector can be calculated in the same manner.

$$E_{conv,j} = \sum_{l=1}^{c_j} \left(E_{mar,l} + E_{unmar,l} \right) \quad \textbf{\textit{Eq. 11}}$$

The facilitation cost $E_{facil,j}$ of client and server connectors is calculated with Equation 12. $E_{remoteConn}$ and $E_{localConn}$ are the energy costs due to establishing a single remote or local connection, respectively. $Num_{remoteConns,j}$ and $Num_{localConns,j}$ are the numbers of remote and local connections established, respectively.

$$\begin{array}{l} E_{facil,j} = \left(E_{remoteConn} \times Num_{remoteConns,j} \right) \\ \qquad + \left(E_{localConn} \times Num_{localConns,j} \right) \end{array} \quad \textbf{\textit{Eq. 12}}$$

By substituting Equations 7 - 12 into the generic energy cost model in Equation 6, we arrive at the client-server energy cost model (omitted here due to space constraints). Note that the energy cost parameters (e.g., E_{toConn}, $E_{fromComp}$, *tEC*, etc.) introduced in this section are platform-specific, i.e., their values depend on the hardware, OS, and middleware on which an application is deployed. We elaborate on how these parameters are determined for an actual platform in Section 3.

2.3 Pub-Sub Energy Cost Model

The pub-sub style reduces coupling by providing transparency and anonymity of component identities and locations. Figure 3 shows an example of a distributed system designed using the pub-sub style. A component in the pub-sub style may be a publisher, a subscriber, or both. Pub-sub constructs have the following characteristics:

Component Communication: Subscribers declare the events they wish to receive by sending subscription requests to a pub-sub connector. They then asynchronously receive the published events of interest from the pub-sub connector. In addition, subscribers can send unsubscription requests to stop receiving certain events. On the other hand, publishers may continuously and asynchronously publish events by transmitting them to a pub-sub connector.

Connector Communication: Pub-sub connectors receive and buffer subscriptions from both subscribers as well as other, potentially remote, pub-sub connectors. Pub-sub connectors also forward subscriptions to other pub-sub connectors, so that they may update their subscription tables. In addition, pub-sub connectors receive and buffer events from publishers and other, potentially remote, pub-sub connectors. They also forward events to subscribers and other pub-sub connectors.

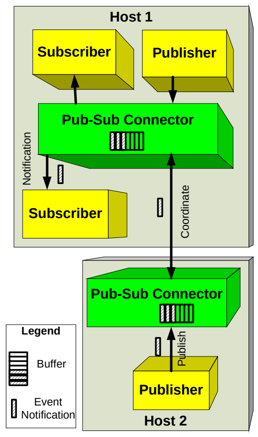


Figure 3. A distributed publish-subscribe architecture.

Connector Coordination: Pub-sub connectors provide time decoupling among subscribers and publishers by queueing events, so that they can be delivered later.

Connector Conversion: Pub-sub connectors marshal/unmarshal events.

Connector Facilitation: Pub-sub connectors (1) manage subscriptions and publications, (2) find the set of subscriptions that match each published event, and (3) create connection objects that implement remote communication.

Based on the above characterization, we can first calculate the energy cost $E_{commWithConn,i}$ of a component $Comp_i$ due to exchanging subscriptions, unsubscriptions, and events with pub-sub connectors as follows:

$$E_{commWithConn,i} = \sum_{k=1}^{a_i} E_{toConn,k} + \sum_{l=1}^{b_i} E_{fromConn,l} + \sum_{m=1}^{c_i} E_{subs,m} + \sum_{n=1}^{d_i} E_{unsubs,n} \quad \textbf{Eq. 13}$$

a_i is the total number of events published by the component, and b_i represents the total number of events received by the component. $E_{toConn,k}$ is the energy cost of sending the kth event to a pub-sub connector, while $E_{fromConn,l}$ is the energy cost of receiving the lth event from a connector. c_i and d_i are the numbers of subscriptions and unsubscriptions sent by the component. $E_{subs,m}$ and $E_{unsubs,n}$ are the energy costs incurred by sending the mth subscription and nth unsubscription to the pub-sub connector.

The energy cost $E_{commWithComp,j}$ of a pub-sub connector $Conn_j$ incurred by exchanging subscriptions and events with components can be calculated as follows:

$$\begin{aligned} E_{commWithComp,j} = & \sum_{k=1}^{e_j} \left(E_{fromComp,k} + E_{recBuffer,k}\right) + \\ & \sum_{l=1}^{f_j} E_{toComp,l} + \\ & \sum_{m=1}^{g_j} \left(E_{rSubs,m} + E_{subBuffer,m}\right) + \\ & \sum_{n=1}^{h_j} \left(E_{rUnsubs,n} + E_{unsubBuffer,n}\right) \end{aligned} \quad \textbf{Eq. 14}$$

e_j is the total number of events received from components, while f_j is the total number of events sent to components. $E_{fromComp,k}$ and $E_{recBuffer,k}$ are the energy costs of receiving the kth event from a components and buffering it, while $E_{toComp,l}$ is the energy consumption of forwarding the lth event to all subscribers of that event. g_j and h_j are the numbers of subscriptions and unsubscriptions received from components. $E_{rSubs,m}$ and $E_{rUnsubs,n}$ are the energy costs incurred by receiving the mth subscription and nth unsubscription, whereas $E_{subBuffer,m}$ and $E_{unsubBuffer,n}$ are the energy costs of buffering the mth subscription and nth unsubscription, respectively.

The energy cost $E_{remoteComm,j}$ of a pub-sub connector caused by sending/receiving subscriptions and events to/from remote connectors can be estimated as follows:

$$\begin{aligned} E_{remoteComm,j} = & \sum_{k=1}^{p_j} \left(\left(rSize_k \times rEC + rS\right) + E_{buffer,k}\right) + \\ & \sum_{l=1}^{q_j} \left(tSize_l \times tEC + tS\right) + \\ & \sum_{m=1}^{r_j} \left(\left(rSubSize_m \times rEC + rS\right) + E_{SubBuffer,m}\right) + \\ & \sum_{n=1}^{s_j} \left(tSubSize_n \times tEC + tS\right) + \\ & \sum_{o=1}^{t_j} \left(\left(rUnsubSize_o \times rEC + rS\right) + E_{UnsubBuffer,o}\right) + \\ & \sum_{p=1}^{u_j} \left(tUnsubSize_p \times tEC + tS\right) \end{aligned} \quad \textbf{Eq. 15}$$

p_j is the total number of events received over the network, r_j is the total number of subscriptions received over the network and t_j is the total number of unsubscriptions received over the network. q_j is the total number of events sent over the network, whereas s_j is the total number of subscriptions and u_j is the total number of unsubscriptions sent over the network. $rSize_k$, $rSubSize_m$ *and* $rUnsubSize_o$ are the sizes of the kth event, mth subscription and oth unsubscription received over the network, respectively. $E_{buffer,k}$, $E_{SubBuffer,m}$ and $E_{UnsubBuffer,o}$ are the energy costs of buffering the kth event, mth subscription and oth unsubscription. $tSize_l$, $tSubSize_n$ and $tUnsubSize_p$ are the sizes of the lth event, nth subscription and pth unsubscription sent over the network. The other parameters are the same in as Equation 9.

We can calculate $E_{localComm,j}$ of a pub-sub connector due to exchanging subscriptions, unsubscriptions, and events with local pub-sub connectors as follows:

$$\begin{aligned} & E_{localComm,j} = \\ & \sum_{k=1}^{p_j} \left(E_{localReceiv,k} + E_{buffer,k}\right) + \sum_{l=1}^{q_j} E_{localTrans,l} + \\ & \sum_{m=1}^{r_j} \left(E_{localSubReceiv,m} + E_{subBuffer,m}\right) + \sum_{n=1}^{s_j} E_{localSubTrans,n} + \\ & \sum_{o=1}^{t_j} \left(E_{localUnsubReceiv,o} + E_{unsubBuffer,o}\right) + \\ & \sum_{p=1}^{u_j} E_{localUnsubTrans,p} \end{aligned} \quad \textbf{Eq. 16}$$

p_j, r_j and t_j are the total numbers of events, subscriptions and unsubscriptions received from local pub-sub connectors, respectively. q_j, s_j and u_j represent the total numbers of events, subscriptions and unsubscriptions sent to local pub-sub connectors, respectively. $E_{localReceiv,k}$, $E_{localSubReceiv,m}$ and $E_{localUnsubReceiv,o}$ are the energy costs of receiving the kth event, mth subscription and oth unsubscription from local pub-sub connectors. $E_{buffer,k}$, $E_{subBuffer,m}$ and $E_{unsubBuffer,o}$ are the energy costs of buffering the kth event, lth subscription and oth unsubscription. $E_{localTrans,l}$, $E_{localSubTrans,n}$ and $E_{localUnsubTrans,p}$ are the energy costs of sending the lth event, *n*th subscription and *p*th unsubscription to local pub-sub connectors, respectively.

The energy cost $E_{coordin,j}$ of a pub-sub connector incurred by performing coordination can be calculated as follows:

$$E_{coordin,j} = \sum_{k=1}^{x_j} E_{queue,k} \quad \textbf{Eq. 17}$$

x_j is the total number of events received by the connector, and $E_{queue,k}$ represents the energy cost due to queueing the *k*th event.

The conversion cost $E_{conver,j}$ of a pub-sub connector can be calculated as follows:

$$E_{conver,j} = \sum_{k=1}^{p_j} E_{unmar,k} + \sum_{l=1}^{q_j} E_{unmarSub,l} + \sum_{m=1}^{r_j} E_{unmarUnsub,m} + \sum_{n=1}^{s_j} E_{mar,n} + \sum_{o=1}^{t_j} E_{marSub,o} + \sum_{p=1}^{u_j} E_{marUnsub,p} \quad \textbf{Eq. 18}$$

p_j, q_j and r_j are the total numbers of events, subscriptions and unsubscriptions to be unmarshalled, whereas s_j, t_j and u_j are the total numbers of events, subscriptions and unsubscriptions to be marshalled, respectively. $E_{unmar,k}$, $E_{unmarSub,l}$ and $E_{unmarUnsub,m}$ are the energy costs of unmarshalling the *k*th event, *l*th subscription and *m*th unsubscription, respectively. $E_{mar,n}$, $E_{marSub,o}$ and $E_{marUnsub,p}$ represent the energy costs of marshalling the *n*th event, *o*th subscription, and *p*th unsubscription, respectively.

Finally, the facilitation cost $E_{facili,j}$ of a pub-sub connector is:

$$E_{facili,j} = \sum_{k=1}^{x_j} E_{route,k} + \sum_{m=1}^{y_j} E_{procSubs,m} + \sum_{n=1}^{z_j} E_{procUnsubs,n} + \left(E_{remoteConn} \times Num_{remoteConns,j}\right) + \left(E_{localConn} \times Num_{localConns,j}\right) \quad \textbf{Eq. 19}$$

x_j is the total number of events received by the connector, whereas y_j and z_j are the total numbers of subscriptions and unsubscriptions received by the connector, respectively. $E_{route,k}$ is the energy cost of retrieving the set of subscribers for the *k*th event from a subscription database. $E_{procSubs,m}$ and $E_{procUnsubs,n}$ represent the energy consumption of processing the *m*th subscription and *n*th unsubscription, respectively. The other parameters are the same as those in Equation 12.

As in the case of the client-server style, the additional energy cost parameters (e.g., E_{subs}, E_{unsubs}, $E_{procSubs}$, etc.) introduced in this section are platform-specific. We discuss how they are determined for an actual platform in Section 3.

2.4 Style-Induced Energy Trade-Offs

To compare the energy costs induced by architectural styles, and illuminate their fundamental differences, we derive the algebraic difference between their energy cost models. For example, by subtracting the equation representing the total energy cost of the pub-sub style from that of the client-server style, a number of terms cancel out, leaving terms representing the costs of connector communication and facilitation. Inspection of these terms clearly reveals that the number of messages exchanged in the two styles is different, and the pub-sub style incurs a facilitation cost that is not present in the client-server style. The intuition behind this result is that the client-server style is based on a point-to-point interaction between components, while the pub-sub style may route messages more efficiently. Therefore, if a message is intended to be received from a remote host by multiple components running on one host, the client-server style in principle requires each message to be transmitted to each recipient component separately. On the other hand, the same could be achieved in the pub-sub style with only one transmission of the message. In this case, it is reasonable to expect the remote communication energy cost of client and server connectors (i.e., $E_{remoteComm}$ of Equations 9 and 10) to be larger than that of pub-sub connectors (i.e., $E_{remoteComm}$ of Equation 15). However, in general, pub-sub connectors incur a higher facilitation energy cost (i.e., E_{facil} of Equation 19) than client and server connectors (i.e., E_{facil} of Equation 12) because the pub-sub style has the additional overhead of managing subscriptions and retrieving them for each published event.

Therefore, through the systematic determination of energy cost parameters, as outlined above, it becomes immediately clear that the energy trade-off between the client-server and pub-sub style is dominated by two factors: (1) the number of separate messages exchanged remotely and (2) the facilitation overhead of a pub-sub connector. In other words, a client-server application consumes more energy than the same application implemented in the pub-sub style only if the energy cost of exchanging additional messages is larger than the facilitation cost of the connectors in pub-sub, and vice versa.

This type of comparison can be conducted for arbitrary architectural styles whose energy costs have been modeled using the process described in this paper. An important contribution of our framework is that it allows architects to intuitively understand the energy trade-offs between different styles based solely on the style-induced characteristics, and irrespective of the implementation platform.

3. ENERGY PREDICTION MODELS

The energy cost models given in Section 2 provide a symbolic representation for assessing the energy cost induced by the architectural style of a distributed system. However, the actual energy consumption induced by various styles is dependent on several platform-specific and application-specific properties. In this section, we show how to apply the energy cost model for an architectural style to a distributed system by creating a specific *energy prediction model*. To derive an energy prediction model, the following information is necessary:

1. *Platform-specific energy cost model parameters* must be determined through mapping the energy cost model parameters to the target implementation platform and measuring their actual costs. These parameters include the cost of transmitting data over a network and the cost of performing lookups in routing tables. The process for performing this task is explained in Section 3.1.

2. *Application-specific energy cost model parameters* must be determined from the system design. These parameters include the system's components, connectors, their configuration, sizes of exchanged messages, etc. The process for gathering these parameters is detailed in Section 3.2.

Note that determining the values of both of the above sets of parameters only requires access to the target platform and basic application design information, and does not require the actual implementation of the application. This feature allows our framework to be utilized by an architect early in the overall design process.

3.1 Platform-Specific Model Parameters

To accurately determine the energy costs represented by platform-specific parameters in an energy cost model, we characterize these costs in terms of interfaces provided by the underlying implementation platform. This process consists of two high-level steps: (1) mapping each platform-specific cost parameter to the interface(s) of the underlying platform that incur that cost, and (2) measuring with a digital multimeter the actual energy consumed when those interfaces are invoked. We have mapped energy cost models to three different platforms: Prism-MW [12], Java RMI [6] and TAO [16]. Due to space constraints, in this section we only describe the mapping for the pub-sub energy cost model onto Prism-MW. The process for other styles, as well as other platforms, is analogous, and the details of these other mappings are given in [18].

Prism-MW, a lightweight, component-based middleware platform, is an appropriate demonstration and evaluation platform for our framework for two reasons: (1) it is intended for resource-constrained and mobile systems, to which our work is directly relevant; and (2) it supports for numerous architectural styles [12], giving us a common platform to evaluate the framework's utility. Prism-MW provides programming language-level constructs that directly correspond to and implement software architecture-level concepts such as components, connectors, topologies, and ports. In general, there are many ways of implementing an interface; in Prism-MW, the invocation of an interface corresponds an event being passed to the *handle* method of one of the Prism-MW constructs. Therefore, we map the energy cost model parameters for the pub-sub style to the Prism-MW platform in the following way:

- The parameters E_{subs}, E_{unsubs}, $E_{procSubs}$ and $E_{procUnsubs}$ (of Equations 13 and 19) are incurred when a pub-sub *connector* handles a subscribe or unsubscribe event.
- The parameters E_{route}, E_{toConn}, $E_{fromComp}$ and $E_{recBuffer}$ (of Equations 13, 14, 15 and 19) are incurred when a pub-sub *connector* handles a published event.
- The parameters $E_{fromConn}$ and E_{toComp} (of Equation 14) are incurred when a *component* handles an event to which it has subscribed.
- The parameters E_{mar} *and* E_{unmar} (of Equation 18) are incurred when a Prism-MW *port* handles an event that must be transmitted over the network. A Prism-MW port is a connection duct for a software connector [12]. Therefore, its energy cost is aggregated with that of the associated connector.

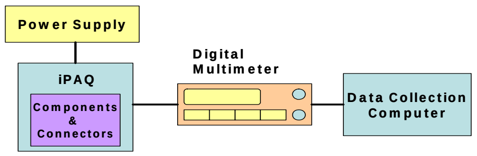


Figure 4. Measurement setup.

Once the energy cost model parameters are mapped to interfaces of the target implementation platform, it is necessary to measure the actual energy cost of invoking each interface, using the measurement setup shown in Figure 4. A benchmarking application invokes each platform interface many times with different input parameter values and measures the current drawn from a power supply. We then apply multiple regression to the values recorded by the benchmark application to estimate the relationship between the input variables and the energy consumed. This relationship depends on the target platform. As one target platform, we used a Compaq iPAQ 3800 mobile device running embedded Linux with a 206MHz Intel StrongARM processor, 64MB memory, and 11Mbps 802.11b compatible wireless PCMCIA card. We chose a version of Prism-MW that runs on top of the JamVM 1.4.5 [7], which is a lightweight Java Virtual Machine. The iPAQ device was connected to an external 5V DC power supply and HP 3458-a digital multimeter. The multimeter sampled the current drawn by the iPAQ at a high frequency. A data collection computer controlled the multimeter and recorded the current samples. Table 1 summarizes the measured energy consumption values.

Table 1. Measured Platform-Specific Parameter Values

Prism-MW Interface	Energy Cost Model Parameters	Average Measured Value (mJ)
Connector. handle(Subcribe)	$E_{subs} + E_{procSubs}$	24.1
Connector. handle(Unsubscribe)	$E_{unsubs} + E_{procUnsubs}$	18.1
Connector. handle(Publication)	$E_{route} + E_{toConn} + E_{fromComp} + E_{recBuffer}$	72.9
Component. handle(Publication)	$E_{fromConn} + E_{toComp}$	40.9
Port. handle(LocalEvent)	$E_{mar} + E_{remoteComm}$	15.0 +2.8**eventSize*
Port. handle(RemoteEvent)	$E_{unmar} + E_{remoteComm}$	12.0 +2.6**eventSize*

Once platform-specific parameter values have been obtained, the only remaining undefined values in the energy cost model are application-specific parameters. The next section describes how to incorporate this information from a system design to create a completely parameterized energy prediction model.

3.2 Application-Specific Model Parameters

Energy cost models contain parameters related to the number and size of messages exchanged between components. Also, whether messages are transmitted locally or over a network has a major impact on energy use. Determining these application-specific parameters requires extracting the following information from the system design:

- The system's constituent components, connectors, and their configuration
- The system's deployment architecture, which is the allocation of the system's components and connectors to its hardware hosts
- Sizes of messages exchanged between components
- A set of destination components for each message

To illustrate how to derive an energy prediction model for an application, given the above information, consider the sensor application shown in Figure 5. The application was designed using two architectural styles: client-server and pub-sub. The *FireAlarmReceiver* component on Gateway 1 translates and aggregates alarms received from fire detection sensors periodically, and propagates them to the *FireAlarmAnalyzer* component on the Hub. Additionally, these alarms are logged by the *AlarmLogger* component. The *FireAlarmAnalyzer* interprets the alarm data to determine whether there is actually a fire. If the *FireAlarmAnalyzer* concludes that there is a fire, it transmits a sensor-activation message to the *FireAlarmReceiver*, which in turn sends an activation signal to all the fire sensors. An analogous processing path takes place in the intrusion detection sensors, *IntrusionAlarmReceiver*, and *IntrusionAlarmAnalyzer* components.

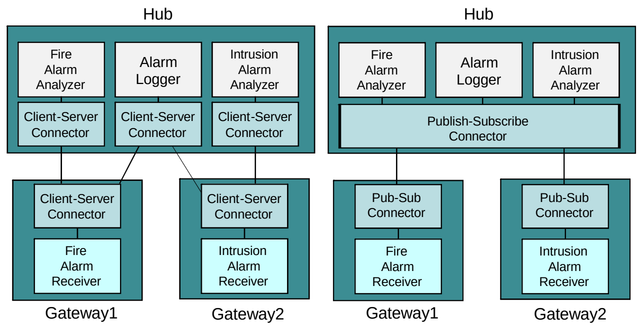


Figure 5. A distributed sensor application designed in client-server (left) and publish-subscribe (right) styles.

In the client-server architecture, the *Receiver* components act as clients and invoke interfaces on the *Analyzer* and *Logger* via their local connectors. The client connectors on Gateways 1 and 2 transmit requests received from local *Receiver* components to the *Analyzer* and *Logger* separately, which indicates that each alarm requires two remote transmissions. On the other hand, in the pub-sub architecture, the *Analyzer* components subscribe to fire or intrusion alarm events, and the *AlarmLogger* subscribes to both event types. When the *FireAlarmReceiver* publishes a fire alarm event, the pub-sub connector retrieves the event's subscribers (i.e., *FireAlarmAnalyzer* and *AlarmLogger*) and routes the event to them, which requires only one transmission from Gateway 1.

Thus, to create an energy prediction model for the pub-sub instance of the above application, an architect characterizes the system in the following way:

- There are five components and three pub-sub connectors, running on three hosts.
- The types of messages are *fire alarm*, *intrusion alarm*, *fire sensor activation*, *intrusion sensor activation*, and *subscription request*.
- The destination components for a *fire alarm* message are *FireAlarmAnalyzer* and *AlarmLogger* components, while an *intrusion alarm* message has the destinations of *IntrusisonAlarmAnalyzer* and *AlarmLogger* components. The destination of a *fire sensor activation* message is the *FireAlarmReceiver* component, whereas an *intrusion sensor activation* has the *IntrusionAlarmReceiver* destination component.

3.3 Style-Induced Energy Trade-Offs

Instantiating a style's energy cost model with platform-specific and application-specific energy cost model parameters leads to an energy prediction model that can be used to estimate the relative differences between styles with respect to energy consumption. For example, in the sensor application scenario, the energy cost incurred on the Gateway by the occurrence of a *fire alarm* event (in mJ) is given in Equations 20 (for the pub-sub style) and 21 (for the client-server style).

$$87.85 + 2.752 * eventSize \qquad \textbf{Eq. 20}$$

$$43.39 + 5.504 * requestSize + 38.07 * 5.128 * replySize \qquad \textbf{Eq. 21}$$

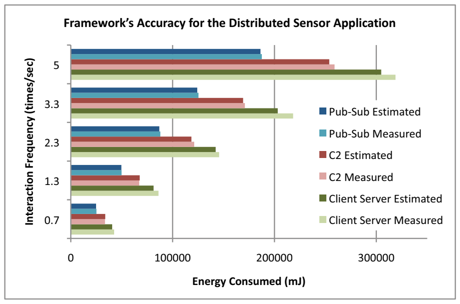
Thus, given rough estimates of the sizes of the various events in the scenario, an architect can determine which style is more efficient. For example, if we assume the sizes of all application events to be 9 KB, the cost of each fire alarm is 112.62 mJ for pub-sub and 177.15 mJ for client-server. If we assume subscription and connection setup (i.e., facilitation) to be one-time costs, over time the pub-sub architecture would be approximately 37% more efficient. More generally, an architect can easily vary any of the parameters in the model — hosts, components, messages sizes, etc. — and investigate which style is most efficient in different situations.

4. EVALUATION

This section presents the results of our framework evaluation. To check the accuracy of our framework's estimates, we measured the actual energy costs induced by the architectural styles for our application scenarios and compared these values to energy consumption predictions made by our framework.

We evaluated our framework using four applications: the sensor application described in Section 3.2, a mobile sales application, a search-and-rescue coordination application, and an XML data streaming application. For each application, we used our framework to estimate the energy consumed on each host, and calculated the overall energy consumption induced by each style by summing up the hosts' energy costs. We then measured the actual amount of energy consumed by the implemented system, again using the digital multimeter setup described in Section 3.2. We varied the frequencies and sizes of data exchanges and component interactions stochastically.

For the sensor application, the framework suggested that utilizing the pub-sub style would result in significant energy savings compared to the client-server style. As shown in Figure 6, the energy consumption estimates from our framework fell within 7% of the measured energy costs for both styles. Moreover, as our framework predicted, the pub-sub style was determined to be much more energy-efficient for this scenario because the energy cost incurred by exchanging more data remotely in the client-server style exceeds the



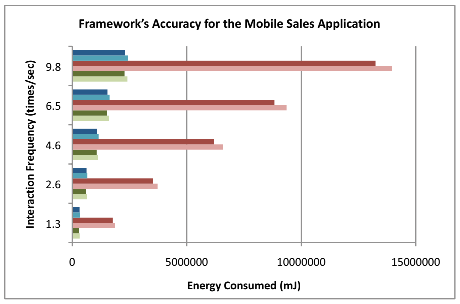


Figure 6. Framework's accuracy for the distributed sensor application (left) and the mobile sales application (right). The values are in *mJ*.

energy overhead due to processing the subscriptions and publications in the pub-sub style.

As shown in Figure 6, the energy consumption predictions for the mobile sales application made by our framework were also within 7% of the actual measured costs for both styles. The framework also correctly predicted that the client-server style is more energy-efficient than the pub-sub style. The same level of accuracy was observed on the other two applications. The detailed data for all applications is given in [18].

5. RELATED WORK

Many approaches have focused on analyzing quality attributes of software architectures (e.g., performance [4] and availability [25]). Likewise, many studies of the characteristics of architectural styles have been conducted (e.g., [13,19]). However, none of these studies focus on energy consumption.

There are a number of tools that estimate the energy consumption of embedded operating systems or applications. Li et al. [10] characterized the energy consumption of the commercial OS, SGI IRIX 5.3 and provided energy consumption models for estimating its runtime energy dissipation. Similarly, Tan et al. [23] investigated the energy behavior of two widely used embedded OSs, μC/OS [8] and Linux, and suggested quantitative macro-models that can be used as energy estimators. Gurumurthi [5] proposed a power estimation tool, *SoftWatt*, using a system simulator, the SimOS, for estimating the energy use of both an application and SGI IRIX 5.3 operating system. Sinha [22] suggested a web-based tool, *JouleTrack*, for estimating the energy cost of an embedded software running on StrongARM SA-1100 and Hitachi SH-4 microprocessors. Flinn et al. [3] developed a tool, *PowerScope*, which estimates the power consumption of mobile applications running on the NetBSD operating system [15] by combining the hardware instrumentation to measure current levels with the kernel software support to perform the statistical sampling of system activities. Recently, Luo et al. [11] suggested a model to estimate the energy cost caused by user interaction with a mobile device. All of the above estimation tools focused on individual applications with concrete implementations and running on specific platforms. Therefore, these estimation tools lack generally applicable energy consumption models.

Several studies [21,26] have characterized the energy consumption of wireless network interfaces on handheld devices. They have shown that the energy usage due to exchanging data over the network is directly linear to the size of data. We leveraged these experimental results to define a connector's remote communication energy cost.

6. CONCLUSION

We presented a framework that facilitates the early estimation of the energy consumption induced by an architectural style on a distributed software system. This capability enables an engineer to employ energy cost predictions along with other quality attributes in determining the most appropriate architectural style for a given distributed application before the implementation of the system. Our extensive evaluation of the framework with respect to accuracy in a large number of distributed application scenarios has shown an error bound of at most 7% as compared to the actual energy cost.

7. REFERENCES

[1] L. M. Feeney, et al. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In Proceedings of IEEE INFOCOM, 2001.

[2] R. Fielding. Architectural Styles and the Design of Network-Based Software Architecture. Ph.D. Dissertation, UC-Irvine, June 2000.

[3] J. Flinn, and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. *Workshop on Mobile Computing Systems and Applications*, 1999.

[4] H. Grahn, et al. Some Initial Performance Characteristics of Three Architectural Styles. In *Proc. of Int'l. Workshop on Software and Performance*, 1998.

[5] S. Gurumurthi, et al. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. In *Proc. of the Int'l. Symposium on HPCA*, 2002.

[6] Java RMI. http://java.sun.com/javase/technologies/core/basic/rmi/.

[7] JamVM 1.4.5. http://jamvm.sourceforge.net/, March, 2007.

[8] J.J.Labrosse.MicroC/OS-II:The Real-time Kernel.CMP Books, 2002.

[9] E. A. Lee. Embedded Software. *Advances in Computers, Academic Press*, London, 2002.

[10] T. Li, and L. K. John. Run-time modeling and estimation of operating system power consumption. In *Proc. of ACM SIGMETRICS*, 2003.

[11] L. Luo, et al. KLEM: A Method for Predicting User Interaction Time and System Energy Consumption during Application Design. In *Proc. of IEEE ISWC*, Oct. 2007.

[12] S. Malek, et al. A Style-Aware Architectural Middleware for Resource Constrained, Distributed Systems. *IEEE Trans. on Software Engineering*, Vol. 31, No. 3, 2005

[13] N. Mehta. Composing Style-Based Software Architectures From Architectural Primitives. Ph.D. Dissertation, USC, 2004.

[14] N. Mehta, et al. Towards a Taxonomy of Software Connectors. In *Proc. of Int'l. Conf. on Soft. Eng.*, Limerick, Ireland, June, 2000.

[15] NetBSD Project. http://www.netbsd.org/, 2005.

[16] D. C. Schmidt, et al. TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems. In *IEEE Distributed Systems Online*, vol. 3, no. 2, Feb. 2002.

[17] C. Seo, et al. A Framework for Estimating the Impact of a Distributed Software System's Architectural Style on its Energy Consumption. In *Proc of WICSA 2008*, Feb. 2008.

[18] C. Seo. Prediction of Energy Consumption Behavior in Component-Based Distributed Systems. Ph.D. Dissertation, University of Southern California, May 2008.

[19] M. Shaw, et al. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. In *Proc. of COMPSAC*, 1997.

[20] M. Shaw et al. Software Architecture: Perspectives on an Emerging Discipline. *Prentice*.

[21] H. Singh et al. Energy Consumption of TCP in Ad Hoc Networks. *Wireless Networks*, 2004.

[22] A. Sinha, et al. JouleTrack - A Web Based Tool for Software Energy Profiling. In *Proc. of Design Automation Conference*, 2001.

[23] T. K. Tan, et. al. Energy macromodeling of embedded operating systems. *ACM Trans. on Embedded Computing Systems*, 2005.

[24] R. N. Taylor, N. Medvidovic, et al. A component- and message-based architectural style for GUI software. *IEEE Trans. on Software Engineering*, Vol. 22, No. 6, 1996.

[25] W. Wang, et al. Software Architectural Analysis - A Case Study. In *Proc. of Int'l. Computer Soft. and Applications Conference*, 1999.

[26] H. Zeng, et al. ECOSystem: Managing Energy as a First Class Operating System Resource. In *Proc. of ACM Int'l. Conf. on ASPLOS*, 2002.