Welcome to the Volume 2 Number 1 of the International Journal of Design, Analysis and Tools for Integrated Circuits and Systems (IJDATICS). This volume is comprised of extended versions of research papers from the 1st IEEE International Conference on Networked Embedded Systems for Enterprise Applications (IEEE NESEA’10) that was held in Suzhou, Jiangsu Province, China in November 2010.

The inaugural NESEA conference provided a high quality forum in which researchers met to discuss the application of networked embedded systems to business processes and the development of technologies that will result in greater application of embedded systems in enterprise scenarios.

This IJDATICS volume presents seven high quality academic papers. This mix provides a well-rounded snapshot of current research in the field and provides a springboard for driving future work and discussion. There are two key themes evident in these papers:

- **Application Design Support**: Four papers investigate how advanced application composition mechanisms can be used to engineer more efficient and flexible networked embedded systems.
- **Efficient Hardware Design**: As networked embedded systems are tightly coupled with the hardware on which they operate, efficient hardware design is essential to a well-engineered system. Three papers tackle this topic.

We are indebted to all of the authors for their contributions to NESEA’10. We would also like to thank the IJDATICS editorial team, which is led by:

Ka Lok Man, Xi’an Jiaotong Liverpool University, China and Myongji University, South Korea  
Chi-Un Lei, University of Hong Kong, Hong Kong  
Kaiyu Wan, Xi’an Jiaotong Liverpool University, China

**Guest Editors:**  
Christophe Huygens, Katholieke Universiteit Leuven, Belgium  
Kevin Lee, Murdoch University, Australia  
Danny Hughes, Xi’an Jiaotong Liverpool University, China

**External Reviewers:**  
Gangmin Li, Xi’an Jiaotong Liverpool University, China  
David Murray, Murdoch University, Australia  
Jo Ueyama, University of Sao Paulo, Brazil
# Table of Contents

**Vol. 2, No. 1, August 2011**

<table>
<thead>
<tr>
<th>Preface</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>ii</td>
</tr>
<tr>
<td>1. Evolving Wireless Sensor Network Behavior Through Adaptability Points in Middleware Architectures</td>
<td>Pedro Javier del Cid, Daniel Hughes, Sam Michiels, Wouter Joosen</td>
</tr>
<tr>
<td>2. Platform Independent, Higher-Order, Statically Checked Mobile Applications</td>
<td>Dean Kramer, Tony Clark, Samia Oussena</td>
</tr>
<tr>
<td>3. Investigation on Composition Mechanisms for Cyber Physical Systems</td>
<td>Kaiyu Wan, Danny Hughes, Ka Lok Man, Tomas Krilavicius, Shujun Zou</td>
</tr>
<tr>
<td>4. Virtualizing Sensor for the Enablement of Semantic-aware Internet of Things Ecosystem</td>
<td>Sarfraz Alam, Mohammad M. R. Chowdhury, Josef Nol</td>
</tr>
<tr>
<td>5. A Low Power and Small Die-Size Phase-Locked Loop Using Semi-Digital Storage</td>
<td>Markus Dietl, Poonet Sareen</td>
</tr>
<tr>
<td>7. Robust Optimization and Reflection Gain Enhancement of Serial Link System for Signal Integrity and Power Integrity</td>
<td>Jai Narayan Tripathi, Raj Kumar Nagpal, Rakesh Malik</td>
</tr>
</tbody>
</table>
Evolving Wireless Sensor Network Behavior Through Adaptability Points in Middleware Architectures

Pedro Javier del Cid, Daniel Hughes, Sam Michiels, and Wouter Joosen

Abstract—Reflection has been proven to be a powerful mechanism to address software adaptation in middleware architectures; however this concept requires that the middleware be open and that modification of all of its functionality and behavior be possible. This leads to systems which are difficult to understand and may quickly overwhelm developers. Safer and more understandable approaches use modeling and put forth a partial implementation of reflective principles while limiting the possible scope of modification, as with translucent middleware. We consider that given the resource constraints in a Wireless Sensor Network (WSNs) it is preferable to limit reflective features in order to conserve computational cycles and reduce network traffic. Additionally we do not believe all modifications lie within the concerns of the application developer and we introduce a separation of operational concerns that maps different modification responsibilities and levels of abstractions to different operational roles. We introduce a middleware architecture that provides strategy-controlled adaptability points; which are available to modify the behavior of the middleware’s primary functionality. We have evaluated our approach through the implementation of a proof of concept prototype that supports an industrial use case in the logistics domain and a need-for-change scenario in the middleware’s capacity planning functionality. Results demonstrate how changes in business requirements may be effectively supported through the introduction of adaptability points.

Index Terms—Middleware, reconfiguration, software adaptation, wireless sensor networks

I. INTRODUCTION

Wireless sensor networks (WSNs) deployments support the integration of environmental data into applications and are typically long-lived, large in scale, resource constrained, subject to unreliable networking and node mobility. In such environments an application needs to adapt its behaviors and functionalities to cope with changing context and operational conditions, by consequence software evolution and reconfiguration become a necessity [1]. Existing approaches mainly focus on extending application functionality or modifying the underlying platform’s execution parameters based on contextual conditions. The use of middleware is a popular approach to address these issues in WSNs [21]; which separate the application from the underlying execution platforms.

Software evolution of WSN applications has been addressed through a variety of approaches e.g. runtime reconfigurable component models [3] and component frameworks [5]. Finer-grained reconfiguration is introduced either through policy based approaches [4] or allowing modifications to code units smaller than components as in TinyComponent [1].

Middleware to allow modification of the underlying execution platform in WSNs commonly use reflective principles, e.g. [2], [19] or partial reflection support, as in [3]. These commonly focus solely on providing the applications finer-grained control over the underlying platform. We currently focus on evolving the middleware itself, specifically in modifying the behavior or the way in which middleware executes its functionality; as opposed to extending its functionality or modifying execution parameters of the underlying platform.

Middleware for traditional distributed systems implement the principle of "information hiding" [15]; which abstracts away implementation specific low-level details and offers higher level abstractions that are simpler to use and configure. In WSNs, given the operational conditions, more control over middleware functionality and behavior is necessary in order to be able to inspect and adapt middleware behavior in favor of optimizing performance [2]. However managing low level details will incur in higher levels of complexity, as is the case of reflective middleware [16]. Reflective middleware makes the internal representation of the middleware explicit and, thus, accessible to be modified; this opposes the principle of transparency or information hiding and through introspection may achieve adaptation. These approaches usually make all functionality and middleware behavior available for modification; which can rapidly become highly complex and difficult to manage [17].

In high power mobile platforms, this increased complexity has been addressed by restricting possible modifications on the middleware and has been approached by enhancing reflective principles with XML based meta-data [17] or multi-layered models of translucent middleware [16].

We consider that given the resource constraints in Wireless Sensor Network (WSNs) limiting the scope of modification is the correct approach but the use of computationally intensive models is not energy efficient.

The Research for this paper was partially funded by IMEC; the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U. Leuven for IWT-SBO-STADIUM [18].

P. J. del Cid, S. Michiels and W. Joosen are with IBBT-DistriNet, Department of Computer Science, Katholieke Universiteit Leuven, Belgium. E-mail: javier.delcid@es.kuleuven.be, sam.michiels@es.kuleuven.be, wouter.joosen@es.kuleuven.be

D. Hughes is with the Department of Computer Science and Software Engineering, Xi’an Jiaotong-Liverpool University, Suzhou, China. E-mail: daniel.hughes@xjtlu.edu.cn
Additionally we do not believe all modifications lie within the concerns of the application developer and we introduce a separation of operational concerns and map different modification responsibilities and levels of abstractions to different operational roles. In this paper we contribute with a middleware architecture that provides strategy-controlled adaptability points; which are available to modify the behavior of the middleware’s primary functionality. Modifying a strategy changes the middleware’s behavior thus modifying how it executes its functionality; in this way changes in business requirements may be effectively supported. To evaluate these capabilities we adapt the middleware architecture that provides strategy-controlled adaptability points; which are available to modify the middleware functionality of our middleware; which we presented and evaluated in [6]. We modify the strategy that controls the capacity planning adaptability point in order to support new business requirements and present the prototype implementation and its evaluation. These new business requirements are introduced in the context of a need-for-change scenario.

This paper is structured as follows: Section II motivates the need to modifying the behavior of the capacity planning functionality. We present the use case, operational roles and the need-for-change scenario. Section III presents an overview of our middleware. Section IV discusses our adaptability points. Section V presents our prototype implementation and its evaluation. Section VI concludes the paper and maps the road ahead.

II. MOTIVATION

In WSNs, functionality commonly addressed through middleware may include: selecting a service provider based on current contextual conditions, modifying sensor sampling frequencies based on available battery, resources, etc. In order to implement the service provider selection functionality, a utility function could be used that accounts for different contextual parameters to rank the suitability of potential providers. One can imagine that in the future, modifications to this utility function may be required for many reasons, e.g. additional contextual sources become available or a more efficient utility function is designed. This gives rise to the need of enacting modifications on how the middleware provides a given functionality, i.e., its behavior, without any modifications to its structure or control flow.

In order to evaluate the notion of adaptability points in middleware architectures, we have implemented a prototype system, made modifications to one of the offered adaptability points in our architecture and evaluated middleware performance before and after the modifications. Specifically, we have modified the runtime capacity planning functionality offered by our middleware. As discussed in Section I, we have presented and evaluated this functionality in [6].

Capacity planning is the practice of estimating the resources that will be needed over some future period of time and is one of the most critical responsibilities in the management of an infrastructure [7]. It is essential to ensure that adequate resources are planned for and provided. Providing runtime capacity planning in our middleware supports the effective control of resource use and enhances system reliability because required resources to process a service request are reserved. This functionality is controlled by a lightweight on-node resource planner. The behavior of which is controlled through a set of strategies. Each strategy is evaluated at a predetermined location in the middleware architecture. These locations are determined based on the importance of the corresponding functionality and the probability that changes to its behavior may be needed in the future. We refer to these locations as "adaptability points". Specifically, capacity planning is controlled by a planning strategy which dictates how and when resources are reserved. This strategy is evaluated at the capacity planning adaptability point.

A. Use case

Our middleware is designed to optimize resource use while considering Quality of Data (QoD) and context aware operation for multi-purpose WSN deployments. In these deployments the infrastructure is considered a light-weight service platform that can provide services for multiple concurrent applications. Concurrently running applications share network resources without inter-application coordination and may have conflicting requirements.

Consider a WSN deployed in a corporate warehouse (see Fig.1). Sensor nodes are deployed at locations A, B, C and D. The deployment is shared by multiple stakeholders, each with its own application requirements. The maintenance department periodically gathers sensing information for a Heating Ventilation and Air Conditioning (HVAC) application. The logistics department deploys a tracking application that provides information on package movement and environmental conditions during shipping of goods.

![Deployment scenario depicted in use case](image)

The HVAC application periodically requests temperature and light measurements throughout the warehouse to determine general AC or heating requirements. Additionally it deploys specialized components to specific nodes that locally determine if an actuating action needs to be taken e.g. if temperature exceeds 30 degrees increase power to the AC unit in this area.

The tracking application monitors Shipping and Handling (S&H) conditions. Warehouse temperature and humidity readings are recorded. On individual packages, position is also monitored. High value packages require light and
accelerometer readings to locally determine package handling and tampering and submit the appropriate alarms when necessary.

Runtime capacity planning in these deployments becomes essential due to the concurrent and uncoordinated use of resources. Consider the common usage pattern in a WSN application, sense-process-react. Successfully supporting this usage pattern requires that the infrastructure is able to provide not only access to the sensor but also provide the memory required during processing, storage and access to the radio to eventually transmit. Additionally one needs to consider that multiple applications compete for limited resources demanding that allocation for these limited resources be done efficiently; thus making the case for runtime capacity planning.

B. Operational roles

In multi-purpose WSNs the main operational concerns involved in application development and use should be undertaken by the following operational roles as defined by Huygens et al. in [13]: application developers, service developers and network administrators. The primary motivation for this separation of operational concerns is based on the fact that managing large scale computational infrastructures across multiple stakeholders is a complicated undertaking. As may be seen from computer networks, web-based service or grid infrastructures. In order to support a large client base and achieve economies of scale in the deployment of such infrastructures a separation of operational concerns is commonly used.

1) Application developers (application owners in [13]) will be concerned with achieving high-level business goals and will undertake the implementation of domain specific business logic.

2) Service developers (component developers in [13]) will be concerned with developing prepackaged functionality to support the goals of the network administrators and application developers. They will undertake the implementation of application-independent and platform-specific common use services e.g. temperature sensing on a SunSpot [14] sensor node i.e. atomic middleware services as later introduced in Section III-A.

3) Network administrators (infrastructure owner in [13]) will be concerned with monitoring network Quality of Service (QoS) and Quality of Data (QoD). They will also configure and maintain common use software services e.g. temperature, aggregation (atomic middleware services as later introduced in Section III-A). They also have high-level goals, usually system-wide requirements driven by concerns such as system lifetime optimization or service level agreements with application stakeholders.

C. Need-for-change scenarios

In this section we put forth two need-for-change scenarios to the capacity planning functionality, in order to exemplify the many situations that may lead to required changes in middleware behavior.

Capacity planning in our architecture is controlled by a planning strategy which dictates how and when resources are allocated and reserved. This strategy dictates how the middleware provides this particular functionality, thus its behavior. Currently this strategy allocates resources on a First Come First Serve (FCFS) basis until the resource’s usage quota is full, after which any additional requests are denied. One may imagine a multitude of situations that would require the modification of this strategy with the intention of changing how and when these resources are allocated. Any of these situations may be regarded as a need-for-change scenario. We elaborate on two scenarios:

1) Prioritizing subscribers: The payment model currently in use for the WSN, is pay per use and does not allow any prioritization of important clients or sensitive data. The FCFS strategy was designed given these considerations. It has been decided that a new payment model will be offered for service usage on the WSN. Different subscription levels will be offered, e.g. elite and standard. Elite subscribers will receive prioritized access to resources and their requests will be processed before standard subscriber requests. Subscriber status should be considered in the planning strategy in order to prioritize resource use. In this scenario elite subscribers are to have priority access to any resource over standard subscribers. Given that the current FCFS planning strategy reserves resources in a first come first serve basis it is not suited to account for subscriber status. This creates the need to modify the behavior of the capacity planning functionality, specifically how the planning strategy allocates resources and what factors are accounted for. Thus a need for change scenario.

2) Compliance with government regulations: New regulations now mandate that all sensor platforms in use in the harbor areas must make their resources available in case of disaster situations, e.g. a fire. In this case sensing and processing data related to the ongoing disaster must take priority over all other allocations. Given that the current FCFS planning strategy does not account for request priorities, this scenario cannot be supported, hence a need to modify middleware behavior. Thus a need for change scenario.

III. MIDDLEWARE OVERVIEW

Our middleware platform is designed to maximize potential resource usage and ensure controlled resource use in multi-purpose WSNs. The workload in this environment is high, concurrent and unpredictable. The middleware actively calculates trade-offs between i) quality requirements associated with service requests and ii) resource capabilities and sensing/actuating alternatives throughout the WSN. Interpretation of these trade-offs enables the middleware to translate service requests to customized component compositions and to instantiate them at well-selected resource providers.

Clients express their requirements through the submission of a service request, in accordance with the service request specification. These requests are parsed and interpreted by a service management layer; which selects the service providers and instantiates a service composition accordingly. We also provide a service framework that defines WSN services and offers mechanism to support concurrency and controlled service use.
A. The service framework

The service framework was designed to present WSN services as a pool of services available to be concurrently used in multiple compositions. It provides support for high loads of concurrent service requests and achieves simpler service composition, fine-grained reconfiguration and higher component reusability. It allows components to be transparently added or removed from any service composition without the need to re-wire existing compositions or interrupt services. Runtime variability in requested QoD can be effectively supported through fine-grained configuration of service compositions. Further discussion on the benefits achieved by the service framework may be found in [9], in the following subsections we provide a high level overview of the framework as is relevant for the context of this paper. The framework defines: 1) service meta-types, 2) service structure, 3) an approach to enable concurrency.

1) Service meta-types: The pool of components available to create service compositions is comprised of basic sensing services and data processing services, these are considered the atomic WSN services. Sensing services (SSC) are components offering typical functionality such as the retrieval of temperature or light readings (see Figure 2). They provide access to the various sensors. Data processing services (DPC) are components implementing post-collection data processing functionality; where the raw sensor data is processed to obtain the desired output.

One may use only one DPC or a pipeline composed of multiple DPCs. In this case, DPCs implement processing steps are connected by data flow through the system, the output data of a step is the input of the following step. Each DPC may enrich input data by computing and adding information, refine data by concentrating or extracting, transform data by producing a new representation, etc. Common processing in WSNs involves, averaging, filtering, calculating a utility function, encryption, etc.

It is important to notice that DPCs may be used to address data qualities in the service request or implement some cross-cutting concerns. For example, temporal aggregation can be achieved with an averaging service, data accuracy may be increased with a specialized data filter that may remove anomalous data values that may indicate a faulty sensor from the raw sensor readings. Additionally sensed data may be prepared and stored in external mediums for persistence or confidential information, e.g. patient data, may be encrypted previous to transmission.

2) Service structure: Components that implement any service in the WSN are provided with typed structure; which is inherited from the service meta-types. All services inherit from a meta-type for which all required and provided interfaces are mandatory. Services may not be extended by adding or modifying existing interfaces unless these changes are implemented at the meta-type level. According to their meta-types, all services inherit annotated attributes. These attributes offer the possibility of encoding runtime accessible semantic information in each service. They may be static or dynamically modified at run-time depending on the attribute and intended use. For example: an energy category attribute is used to represent energy consumption incurred in the invocation of a particular sensor, given that energy use may vary considerably by platform / sensor hardware as exemplified in [10]. The annotated attributes selected for runtime modification by the adaptation interface are also enforced by the meta-type. E.g. sampling frequency attribute in SSCs is modified at runtime by the middleware based on battery level.

Additionally all SSCs must implement the Singleton pattern [12]. It is also required that timestamps are included for every sample of raw sensor data to improve data accuracy. Component coordination and interaction patterns are dictated by the underlying component model.

3) Enabling concurrency: Concurrent use of services in our framework is achieved through reuse of component instances. Given the intrinsic resource constraints in WSNs dealing with service contention through the replication of component instances is not an efficient approach; for this reason, we introduce a configuration meta-level on top of components. We separate a component’s functional code from its meta-data and share the same component instance across multiple service compositions (see Fig. 3). This meta-data contains the configuration semantics to be used in each composition in order to support the client required QoD.

Examples of meta-data for SSCs include corresponding request Id, sampling frequency and duration of service. In the DPCs one may use: request Id, parameter and source Id. The parameter is used by the DPC to parameterize its functionality, in the case of the averaging DPC this determines the time window for the average, i.e. average every 60 min.
Each component is associated with a particular service composition through a request Id; this association contains per-instance configuration semantics. Configuration semantics for each service composition are extracted from the client specified service request. The configuration semantics include client specified QoD, services involved in each composition and related parameterization.

This allows a single instance of our components to be used across multiple service compositions with varying parameters in each composition and avoids substantial increases in required static and dynamic memory per additional service request because only one component instance is instantiated per service type for multiple requests.

B. The service request specification

Clients use the service request specification to express their QoD requirements in a per-service instance manner. We consider a service instance to be: each service request from the moment it is submitted to the middleware until it has been processed as specified. A client or application using the WSN may have multiple concurrent service-instances, e.g. sense temperature and humidity at warehouse X every 15 minutes for the next 3 days.

In the specification one expresses the request Id, which is a unique sequential number generated by the WSN backend middleware. The service Id represents a globally unique service identifier defined at service implementation. Each sensing service e.g. temperature, humidity, has a unique service Id. The temporal resolution required from the specified service is expressed through the sampling frequency. Duration of service i.e. the amount of time one requires the selected sensing service to collect data samples. Spatial resolution is specified by selecting a target location e.g. <warehouse A> or <node21>. A data processing service Id, which is globally unique identifier for services like averaging or specialized data filters. Every data processing service requested requires a parameter be specified for configuration, e.g. in case of the averaging component, one may use the parameter 30 to indicate the average must be done in 30 minute intervals. Each service request may be configured with different QoD requirements and it may or may not include one or more data processing services. Optionally a status may be included to allow the middleware to customize parsing of the service request.

Listing 1: Service request format:

```java
dataRequest#(requestId,serviceId, samplingFrequency,duration,targetLocation,DataProcessServiceId[],parameter[],status);
```

Per-service instance configuration allows multi-purpose WSNs to serve different types of applications with arbitrary requests or query patterns with no a-priori knowledge needed. They provide application developers the flexibility to meet variable QoD requirements of new applications and yet expect the same levels of performance that would result from an application-specific deployment [8]. Fine-grained optimization is possible because every instance may be customized with specific QoD requirements allowing for higher component re-usability, more efficient parameterization and improved reliability through lightweight run-time capacity planning [6].

C. Autonomic service composition

Service composition involves the definition of the processing order and configuration of service interaction in accordance to the client specified service request.

Valid service compositions: Service compositions can have only 1 SSC and zero-to-many DPCs (see Fig. 4). Multiple SSC are not allowed and all DPCs must be configured in sequence. Compositions must follow the pipe and filter pattern [11]. We extend the pattern to also allow for batch processing, where a component may consume all the data before producing an output; as opposed to only consuming and delivering data incrementally.

![Fig. 4. Valid service compositions](image)

This definition appears rather simple but it is capable of representing a wide range of service compositions in multi-purpose WSNs. For example: i) sense temperature, ii) sense and average humidity iii) sense, average, encrypt light, iv) sense, filter, persist methane, V) sense, encrypt, reliably transmit temperature. As one may see this definition is capable of capturing important functional, data quality and cross-cutting concerns.

However, this definition does not cover the composition of composite services i.e. services that require multiple inputs. For example: assessing risk of fire; where temperature and light readings are used to calculate the probability of a fire starting in a given area. As one may see, this composition violates the definition because it has two source components providing input to a filter. We consider that these services should be addressed with the implementation of application specific components; which are considered as consumers or clients in our model. Logically one may assume that in turn they may be considered as services by other components or applications higher above the abstraction level. As is the case of the S&H tampering component introduced in the use case Section II-A.

The service composition process: it begins with the submission of a service request to the service management layer; which is accessible through a Service Management Component (SMC). It automatically interprets requests, selects the optimal service providers and instantiates an individual service composition involving specified services from a shared pool of components interacting in a loosely coupled manner. Every application/client may submit multiple service requests, each representing a service instance. As such, every composition allows for per-service instance parameterization of how this pool of components is
used. In this way, requirements from different users are handled independently, thus avoiding potential conflicts due to resource competition or varying QoD requirements.

![Sequence of processing steps to achieve a service composition.](image)

1) **Parse request**: service ids, their configuration parameters and duration of service are extracted.

2) **Analyze composition**: Parameters of each service requested are verified within the context of the requested composition. For example: one cannot average data samples within a time period smaller than the sampling frequency of the raw sensor data, i.e. you need at least two raw data samples per average interval. As one can see the validity of the parameter for the average service varies depending on the other requested services.

3) **Evaluate providers**: A potential set of candidate nodes is generated based on matching of target location and availability of required services. Service matching is done syntactically; given all services have a globally unique event Id. This event Id is generated according the event type hierarchy presented in [3] and assigned when each service is implemented. These candidates are evaluated given their currently offered data quality properties, battery level, node load, etc.

4) **Select Provider**: The evaluation made in the previous process guides the node selection strategy for the selection of service provider.

5) **Capacity planning**: Required indirect resources are calculated based on the configuration parameters in each service request. The corresponding resource reservations and allocations are fulfilled by the capacity planning functionality of our middleware platform. Further details on capacity planning are provided in Section IV-A.

6) **Create Composition**: The service management layer uses the configuration parameters extracted from the service request to configure the requested services, creating a service composition (see Section III-A). As one may recall, these services may include sensing services and data processing services.

### D. Controlling resource use

Physical resources are exposed though the use of services. Services control the invocation of actual sensors, generation of data or use of any other underlying physical resource, e.g. memory, processor, network, etc. These services are guided by and controlled by the middleware. Clients can only submit their service request, where their usage requirements are specified, to the middleware but exert no direct control over any of the resources. For example a client may request temperature sampling every 10 seconds, this request may be accepted or rejected based on the maximum sampling frequency currently offered by the temperature service in the corresponding sensor node but the client has no control over this maximum.

Our middleware platform controls resource use with the use of two mechanisms: capacity planning and localized adaptation. Capacity planning ensures that only service invocations that are within current permissible usage parameters are allocated to be processed. The capacity planning process estimates the resources required to support a service request and checks availability of each required resource. Usage quotas per resource are used to specify how much of a given resource may be allocated for each activity.

Localized adaptation is an autonomic and independent process guided by adaptation strategies. These adaptation strategies are designed to evaluate how often a resource e.g. sensor, may be used under current system conditions and still maintain quality requirements. For instance, given a battery level of 25%, power hungry sensors may only be invoked once every 10 minutes. These strategies are evaluated locally at node level and directly modify component parameters that limit the use of each resource accordingly. As demonstrated in [10] controlling frequency of invocation of high power sensors significantly lengthens node lifetime. Additionally, the implementation of the singleton pattern [12] in all SCCs provides effective support for resource control.

### IV. ADAPTABILITY POINTS IN OUR MIDDLEWARE

Proposed design principles for adaptive applications have steered application development to implement functionality in a modularized fashion such as inspired by component based engineering principles. In these approaches, formal interfaces are exposed to allow for component parameterization and the modification of functionality [20]. Of course finding the appropriate extent of modularization and determining its impact on performance are important issues. Furthermore, it is generally considered that modification of any modularized part of the application lies solely in the responsibility of a single operational role and that this single operational role has advanced knowledge of the hardware platform, execution platform, middleware and domain specific application software.

We have implemented our middleware functionality in a modularized fashion in such a way that modifications to these modularized portions may be offered to different operational roles and at different levels of abstraction. It is for this purpose that we separate what the middleware does, i.e. its functionality from how it does it, i.e. its behavior. The functionality is modularized with the use of components. The behavior is separated from functionality and evaluated within strategies. The locations in which strategies are called upon and evaluated within components are called adaptability points.

It is important to notice that the abstraction level at which modifications are made to components and strategies may vary significantly. The implementation and modification of components requires knowledge of the middleware, for example: the component model in use, coordination model and underlying execution platform. The implementation or modification of a strategy requires understanding of the different adaptability points available within the architecture.
and knowledge of how to express the desired behavior in a strategy. One may use event condition action semantics to express the desired behavior within a strategy.

Essentially, the logic that controls how primary functionality is executed, i.e. its behavior, has been externalized through the use of strategies. These strategies are called upon during runtime to guide the execution of component functional code. For instance, every time a service request in a sensor node is received, the planning component functional code. For instance, every time a strategy is called upon to evaluate if there are enough resources available to support the request and to allocate resources if necessary. An adaptability point refers to a location where calls to strategies are made within the execution of functionality. The capacity planning component contains the corresponding variability point, where the planning strategy is called upon. We have augmented all the core middleware functionality with strategies as to allow network administrators to enact behavioral adaptations without the need of advanced knowledge regarding the component implementation, underlying runtime environment or hardware platform.

It is important to note the clear distinction between the extension of functionality and the adaptation of its behavior. The former refers to adding new functionality, for example if we include support for component deployment in the middleware platform. The later refers to modifying the decision logic that guides the functionality, as per our need-for-change scenarios, where the current planning strategy will need to be changed to account for emergency situations by allocating emergency service invocations to the corresponding memory quota. Hence the decision logic that decides to grant resources or not for an invocation is modified but all the mechanisms that calculate requirements and later reserve them remain unchanged. It is for modification of behavior only that we have included these adaptability points in the architecture. We address extensibility through predefined plug-in locations in the architecture but these are outside the scope of this paper.

In the broader context of software engineering one may find direct resemblance between adaptability points and joint points in Aspect Oriented (AO) approaches. They both may indicate points in component code where an external construct is called upon to aid execution, aspects in AO and strategies in our approach. A similarity that is only relevant from an implementation perspective. At a conceptual level an aspect is fundamentally introduced to deal with cross-cutting concerns like logging or persistence. Their implementation, maintenance and deployments is considered to be undertaken by the same operational role and does not support a separation of operational concerns.

At a conceptual level, our work is more closely related to the levels of abstraction presented in the PERPOS middleware [16], as these too rely different abstraction levels to aid adaptation efforts. The use of strategies is certainly not new, as they are a subset of the broader policy concept. The main contribution of our works lies in that adaptability points are introduced to aid middleware evolution by allowing adaptation to be done at different abstraction levels and by different operational roles.

In this section we provide further details on the adaptability points corresponding to the capacity planning functionality and the corresponding views and different levels of abstraction mapped to their respective operational role. The different operational roles have been previously introduced in Section II, the application developer, the service developer and the network administrator.

### A. Capacity planning process flow

Capacity planning functionality was designed to plan for and reserve consequential or indirect resources, thus avoiding consequential contention. Direct contention over a resource is currently managed through the use of low level mechanisms commonly offered by the underlying operating system (OS) or virtual machine (VM). For example, components A and B need access to a single light sensor and corresponding CPU cycles (see Fig. 6).

A consequential resource refers to any non-direct resource needed to support an allocated request e.g. when using a sensor you will need not only access to the sensor itself but dynamic memory for processing and static memory to store the data or access to the radio to transmit. We define consequential contention as the moment when two or more running services require a limited consequential resource e.g. memory, to accomplish their tasks and there is not enough availability to serve these requests appropriately.

Capacity planning starts after the service provider has been selected (see Fig. 5) and it consists of two main processes. Mainly, calculating resources needed and reservation of resources (see Fig. 7). These are initiated every time a service invocation is submitted to any service. These invocations for services are accompanied with all the relevant configuration parameters; which are used by the resource planner to calculate indirect resources required. In the case of SSCs these parameters include sampling frequency and duration of service.

![Sequence of processing steps required for capacity planning.](image)

1) **Calculate resources:** In order to effectively calculate the amount of static and dynamic memory that will be used by the middleware we use an off-line process to establish a memory baseline and a run-time process to establish run-time memory requirements. Each node has a light-weight resource planner that is able to estimate specific amounts of memory needed to fulfill each request for both SSCs and DPCs.
2) Reservation of resources: The resource reservation process is mainly comprised of two main sub-processes: and offline calculation of each component's resource use and a runtime capacity planning process. The offline process is executed by the service developer every time a new implementation for any service is developed and the runtime process is executed autonomously by the resource planner during system execution. The resource planner reserves these required resources. This guarantees that every service request will have the needed consequential resources e.g. memory, to be processed successfully through the service duration. \textit{The reservation is done on a first come first served basis.}

Off-line, a Max usage quota for each memory medium should be defined. The network administrator should define what portion of available dynamic (RAM) memory should be used to support running services and what portion of static (flash) memory should be used to support persistence for the generated data. For example: if the total available amount of RAM is 10 Kb, allocate 50% to support running services. These memory usage quotas represent the 100\% of dynamic and static memory that is available for the resource planner to allocate. Every reservation granted subtracts from the memory quota. Every resource that is released after service duration time expires adds back to the amount of memory available from the corresponding quota.

B. Sub-processes of the calculate resource process

This process consists of two sub-processes: generating a baseline of component resource use and runtime capacity planning.

Generating a baseline for component’s resource use: A baseline of resource consumption is recorded for each component type on each platform i.e. hardware and runtime environment, to be used. This process is done off-line. The baseline includes:

1) Static memory requirements to store component code: record the memory required to store component executable code in flash memory.

2) Dynamic memory requirements: Each component requires dynamic memory to be instantiated. It also requires varying amounts of memory during the execution of its functional code and the creation and maintenance of the configuration meta-data (see Fig. 4). At different moments in time during these processes, the amount of required memory varies. To account for this, we collect memory usage information at various points in the processing cycle. This provides the max peak of memory consumption per complete processing cycle. We consider these max peaks of required memory will be the same for any request processed in a particular service type. Each specific SSC or DPC must be measured accordingly.

During the processing of multiple service requests we measure the dynamic memory required to serve additional service requests in a component instance and any additional memory required due to housekeeping overheads such as lagging garbage collection.

Additionally, random measurements are taken while storing records of varying sizes from 32 byte to 32Kb. These are indexed with a request Id, in both dynamic and static memory. In this manner we ascertain the amount of memory overhead generated by platform specific data storage and related housekeeping.

The total required bytes of static and dynamic memory (see Fig. 8A) are recorded and assumed to be constant for a particular implementation of each specific service type. Every component has these amounts recorded as values available and introspect-able as annotated component attributes (see Fig 8B).

Run-time capacity planning: Each service composition specifies which components will be used to serve a specific service request. For every component to be used, we calculate both dynamic and static memory requirements; even though it is possible that the static memory will only be used after the last component has finished processing the data sets. We do this because a memory management strategy specifies that when battery level is under 10\%, the persistence service stores all data sets from all running services to ensure no data-loss. For this reason we need to assure that enough static memory is available for any data set being processed.

The on-node resource planner performs run-time calculations to determine exact amounts of memory required to process each service request according to the meta-data provided for configuration. During processing in an SSC or DPC the entire record set is kept in dynamic memory and transferred to static memory by the persistence component only after the data set has been processed. Sensed or processed data available in dynamic memory is only erased after persisted to static memory or through a specific delete instruction available in the data retrieval interfaces of the SMC, DPCs or SSCs. Calculations for SSC and DPC are different and described below.

1) Calculations for the SSC: Dynamic memory required is equal to the output data set size plus the dynamic storage overhead. The Static memory required is equal to the output data set size plus the static storage overheads. The calculation of output data set size must be done at runtime for each request because the amount of records produced will vary depending on sampling frequency and service duration. Given the amount of records and the data type we can calculate the output data set size. Storage overheads are known from the off-line baseline.

2) Calculation for the DPC: Required dynamic memory is equal to the input data set size plus the dynamic storage overhead plus the output data set size plus the dynamic storage overhead. Required static memory is equal to output...
data set size plus the static storage overhead. To calculate the size of the input dataset: record count is obtained from the data and multiplied by the data type size. The size of the output data set varies depending on the mathematical function applied to the samples and the specific configuration parameters. In the case of the averaging component and other time-series analysis based components: i) time-span incurred in the time samples is calculated based on the timestamps on the data. ii) \([\text{time-span}] / [\text{user specified parameter}]\) = amount of records the output data set will have. We multiply the amount of records and the byte size of the data type, this gives us the required memory for the output dataset.

In the case of filtering components we assume that in the worst case the output data set will have the same size as the input data set. This is done because there is no way to predict how many data samples will be filtered out, but we know no records will be added, so estimating equal input and output data sets is a safe assumption. To these values we add corresponding housekeeping overheads.

The memory reservation is valid for the middleware layer only. This means that service requests are only allocated to components when the consequential resources required are available.

D. Adaptability points in the capacity planning functionality

Adaptability mechanisms in our architecture are mapped to the corresponding operational role. We consider that given the scope of each operational role the appropriate level of abstraction varies. Therefore we offer three different levels of abstraction specifically designed for the scope of adaptation corresponding to each operational role (see Fig. 10). In order from most to least abstract, these are: the client layer, the adaptation layer and the deployment layer.

C. Sub-processes of the reservation of resources process

The resource reservation process is comprised of two alternative sub-processes. Mainly, a reservation for each component instance and a calculation for each service request.

For each component instance: These calculations are updated in two scenarios: i) A new component is going to be instantiated. ii) A component is removed from the node. As one may see in Fig. 9B, the reserved static memory is equal to the required static memory as calculated in the offline procedure (see Fig. 8A). The reserved dynamic memory is equal to the required dynamic memory as calculated in the offline procedure (see Fig. 8A). For every service request: The reserved static memory is equal to the run-time static memory calculated during run-time capacity planning (see Fig 9B). The reserved dynamic memory is equal to the run-time dynamic memory calculated during run-time capacity planning and the dynamic memory required for additional services as calculated in the offline process (see Fig 9B).

An example of run-time capacity planning and resource reservation for SSCs and DPCs is shown in Fig. 9.

1) Client layer: Application developers interact with the middleware API only; they are abstracted away from all underlying layers. However, they can use the API to access different hierarchical levels in the network architecture. They have access to the service management layer, which is present in the backend middleware and the cluster heads (see Fig. 11). The API also gives them access to services on sensor nodes directly, configuring SSCs and DPCs directly. Even at node level they are abstracted away from the low level details. Even while abstracted away from middleware and platform details, the application developer may considerably customize the WSN and extend it using application specific functionality. The deployment of application specific components into the network allows for higher degrees of interaction, in-network processing and localized actuation. These components can easily exploit the WSN services using the configuration method calls offered for SSCs and DPCs in the node level API.

2) Adaptation Layer: Network administrators look at the multi-purpose WSN in terms of processes and available strategies where they may modify the behavior of middleware functionality to meet new business requirements (see Fig. 12), as exemplified in Section II. Needless to say there is always an important balance to be maintained between the degree of modifiability an architecture offers and
its runtime performance. Therefore adaptability points are only available for the primary functionality of our middleware. As is to be expected, some changes in business requirements will require middleware extension and structural adaptation, which are outside the scope of the administrator’s responsibility.

3) The deployment layer: At this level of abstraction the service developers deal with platform specific details and implementing WSN services. It is at service development time that adaptability points are implemented. These adaptability points mark locations in the component code, where strategies are called upon and evaluated. Middleware extension and structural adaptation happens at this layer. Behavioral adaptation may also be executed at this layer by modifying all decision logic that is hard coded into the components or by modifying adaptability points.

V. IMPLEMENTATION AND EVALUATION

In this paper we presented a middleware architecture that provides strategy-controlled adaptability points to enable the modification of middleware behavior. We introduced a use case and need-for-change scenarios in Section II. We have discussed how modifying a strategy changes the middleware’s behavior thus modifying how it executes its functionality and, in this way, changes in business requirements may be effectively supported. To evaluate these capabilities we adapt the capacity planning functionality of our middleware; which we presented and evaluated in [6].

1) Previous prototype implementation to support the presented use case: In [6] we implemented and evaluated a prototype of our middleware platform that supports the use case presented in Section II. To provide the reader with some useful information regarding that evaluation, we describe it further.

In that evaluation we submitted a total of 20 service requests in 6 successive and overlapping batches to the SMC as depicted in Fig. 14, requests from 1 to 20. The x-axis depicts elapsed time and the y-axis depicts the request Ids. One can see the submission times and durations of all requests, in Fig. 14 as they were submitted and Fig. 15 as they were actually processed. As one can see requests 6,7,13,14 and 15 were rejected by the system.
memory under the specified MaxQuota. Our results demonstrate that the consequential resources needed to support concurrent service requests. Memory is reserved to guarantee all allocated services are supported and released after it is no longer needed.

![Graph showing RAM or dynamic memory usage for the specified use case.](image)

Fig. 16. RAM or dynamic memory usage for the specified use case

In Fig. 17 one may see how the actual static memory used is very low compared to the requested and reserved amounts. When instantiating a composition static memory is requested for all components, in this case the temperature and averaging components have static memory reserved but only the persistence component actually uses it. This is because there is a system policy that requires all components processing a service to have enough static memory available in case battery levels drops under 10%. Further elaboration on how modifying this system policy may affect the resource reservation is outside the scope of this paper.

![Graph showing Flash or static memory usage for the specified use case.](image)

Fig. 17. Flash or static memory usage for the specified use case

2) Supporting changes in business requirements through adaptability points in middleware architectures: As previously mentioned, in order to evaluate these capabilities we need to modify the planning strategy that controls the capacity planning adaptability point. This strategy was originally designed to allocate resources based on a first come first serve decision logic. In order to support the need-for-change scenario: Compliance with government regulations, the strategy needs to be modified to support resource allocation based on service request priorities.

Previously the planning strategy would reject any new request when the memory quota was full. If an emergency situation where to happen while the quota was full, it would have been rejected as any other request. We have now modified this strategy to allow emergency requests be allocated even after the service quota is full. We have implemented an emergency quota for emergency use reserved only for emergency situations, thus providing support for these even after the normal memory quota is full; in accordance with the need-for-change scenario.

Service requests are now parsed to determine if they are normal requests or emergency requests. The resource planner now proceeds with capacity planning accordingly, based on service request priorities.

Furthermore it is important to notice that as previously discussed, the modification of the planning strategy is within the responsibility of the network administrator. Modifications required to support this need for change scenario are enacted within the adaptation layer and achieved at the level of abstraction suitable to the expertise of a network administrator.

This demonstrates that changes in business requirements may be supported by network administrators at a convenient abstraction level. It also shows that the separation of operational concerns into corresponding roles and mapping adaptation responsibilities to each; is a feasible approach to evolving middleware behavior in the domain of multi-purpose WSNs.

![Graph showing Requests as submitted to the system after modifying the planning strategy.](image)

Fig. 18. Requests as submitted to the system after modifying the planning strategy

![Graph showing Requests as actually processed after modifying the planning strategy.](image)

Fig. 19. Requests as actually processed after modifying the planning strategy

As one may see in Fig.18 we have submitted a total of 20 service requests in 6 successive and overlapping batches to the SMC, requests from 1 to 20, as we have previously done. Additionally, emergency requests 21 and 22 are submitted. The x-axis depicts elapsed time and the y-axis depicts the request Ids. One can see the submission times and durations
of all requests, in Fig. 18 as they were submitted and Fig. 19 as they were actually processed. As one can see, now emergency situations are effectively supported. At t4 in Fig. 19 one can see that even under full load condition emergency request 22 is supported and requests 13, 14, 15 are rejected.

Figures 20 and 21 illustrate RAM and FLASH usage during the processing of these requests. As one may see that for emergency request 22 at t4, the RAM usage has exceeded the MaxQuota, but since we have allocated additional memory space exclusively for emergency situations, the system may still effectively support the emergency request while rejecting a normal request.

Fig. 20. RAM memory usage during the processing of these requests

Fig. 21. Flash memory usage during the processing of these requests

As one may see, the planner still maintains allocation controlled and as before, every request that exceeds availability is rejected, while allowing emergency requests to be processed. In Fig. 21 one may see how the actual FLASH memory used is very low compared to the requested and reserved amounts. When instantiating a composition static memory is requested for all components, in this case the temperature and averaging components have static memory reserved but only the persistence component actually uses it. This is because there is a memory management strategy that requires all components processing a service to have enough static memory available in case battery levels drops under 10%. Further elaboration on how modifying this strategy may affect the resource reservation is outside the scope of this paper. All memory measurements provided are obtained by using the memory management facilities offered by the Sunspot API [14].

VI. CONCLUSION

In this paper we have presented a middleware architecture that provides strategy-controlled adaptability points; which are design to enable network administrators to enact behavioral adaptation in the middleware's functionality in order to support changes in business requirements. We presented three operation roles that represent a separation of operational concerns. We have mapped each operational role to a different abstraction level corresponding to the scope of modification responsibilities of each role. To evaluate these capabilities we introduced a need-for-change scenario, which we have implemented and successfully supported.

In this manner, we have demonstrated that changes in business requirements may be supported by network administrators at a convenient abstraction level. It also shows that the separation of operational concerns into corresponding roles and mapping adaptation responsibilities to each; is a feasible approach to evolving middleware behavior in the domain of multi-purpose WSNs.

In our future work we plan to evaluate this functionality under an extended set of use cases that prove the applicability of the approach beyond logistics scenarios. We will prepare a hybrid testing infrastructure composed of a physical network and simulated environment to evaluate the efficiency of these strategies under high workloads and dynamic conditions.

REFERENCES


[14] SunSPOT: http://www.sunspotworld.com/, visited July 2010


Pedro Javier del Cid received an engineering degree in electronics degree from Universidad Francisco Marroquin, Guatemala in 2000. He received an advanced masters degree in artificial intelligence from Katholieke Universiteit Leuven in 2001. He is currently a researcher at IBBT-DistrinNet and a PhD candidate in computer science at Katholieke Universiteit Leuven. His research interests include adaptive middleware for pervasive systems and architecture for software intensive systems.

Sam Michiels received a M.Sc. in informatics and a Ph.D. in computer science from Katholieke Universiteit Leuven in 1998 and 2003 respectively. He is currently Research Manager (Industrial Research Fellow, K.U.Leuven-IOF) within the DistriNet research group of the K.U.Leuven; this mandate enables him to focus on research, innovation and especially technology transfer in the domain of managing large-scale distributed software systems. His current activities target sensor networks and the design of adaptable middleware.

Wouter Joosen is full professor at the Department of Computer Science of the Katholieke Universiteit Leuven in Belgium, where he teaches courses on software architecture and component-based software engineering, distributed systems and the engineering of secure service platforms. His research interests are in aspect-oriented software development, focusing on software architecture and middleware, and in security aspects of software, including security in component frameworks and security architectures.

Danny Hughes received a B.S. in computer science, a M.Sc. in distributed interactive systems and a PhD in adaptive peer to peer systems at Lancaster University, England in 2001, 2002 and 2007 respectively. His research interests include: wireless sensor networks, middleware, embedded systems, distributed systems and peer-to-peer systems. He is currently a lecturer and faculty member of Xi’an Jiatong-Liverpool University, Suzhou, China and a free associate with Katholieke Universiteit Leuven, Belgium.
Platform Independent, Higher-Order, Statically Checked Mobile Applications

Dean Kramer, Tony Clark, and Samia Oussena

Abstract—There is increasing interest in establishing a presence in the mobile application market, with platforms including Apple iPhone, Google Android and Microsoft Windows Mobile. Because of the differences in platform languages, frameworks, and device hardware development of an application for more than one platform can be a difficult task. In this paper we address this problem by the creation of a mobile Domain Specific Language (DSL). Domain analysis is carried out using two case studies, leading to the identification of basic requirements for the language. The language is defined as an extension of a $\lambda$-calculus in terms of an operation semantics and a type system. The language is a contribution to the understanding of mobile applications since it precisely defines the essential properties offered by a range of mobile application technologies, and can form the basis for a single language that can target multiple platforms.

Index Terms—Domain Specific Languages, Mobile Computing, Platform-Independence

I. INTRODUCTION

TODAY, the penetration of modern smart phones is vastly increasing with over 172 million smart phones shipped worldwide in 2009 [1], and with the emergence and successes of sources for consumers to install third party applications opens a new market for developers to reach consumers. However, developing an application for multiple mobile platforms can incur different obstacles including differences in development tools available, different languages, platform constraints and availability of software libraries.

Difficulties in producing software for more than a single platform has been evident for many years outside of the mobile realm. For decades, software portability has been a concern during development, mainly due to very large spectrum of different CPU Instruction Set Architectures (ISA) and by the large variety of Operating Systems in use. Recently this has become less of an issue, largely due to factors including the decrease in CPU ISAs, the dominance of a limited number of operating systems, and to commonly used languages including Java. Since the mobile market is relatively immature, there are large differences in implementation languages and development environments used for different applications and technology platforms.

Software porting and cross platform development remains the most common method for multi-platform development. For large software companies this is not an issue, but for smaller commercial mobile application businesses this presents a problem. Firstly, each mobile platform has a different implementation language and therefore each developer needs to be multi-lingual or a company needs to set up multiple development teams for each product. Secondly, businesses must invest in multiple types of testing equipment for the different platforms. These factors lead to an economic driver for technology that can help to deliver mobile applications over families of target platforms.

A. Outline of Paper

The outline of this paper is as follows: section II describes the background to mobile application development; section III outlines current approaches to multiplatform methods and defines our contribution; section IV describes two case studies that we have implemented for a company and which leads to the identification of the domain features for our language; section V defines the syntax of the language and gives a simple example application; section VI defines the language executes; finally section VII analyses the language and describes our current and future implementation strategies.

II. BACKGROUND AND RELATED WORK

A Software Development Kit (SDK) provides the environment for developing applications through the use of libraries, emulators, and debuggers and is the basis for the development of the majority of current mobile applications. However an unfortunate drawback of SDKs is that they are platform dependant: Apple provides an Xcode SDK (http://developer.apple.com/devcenter/ios) that includes an interface builder, an iPhone simulator and development environment; the Android SDK comes as an eclipse plug-in, and includes a software emulator; the Windows Phone provides a specialised version of the Microsoft Visual Environment; Blackberry and Symbian also provide different SDK platforms. This variance in terms of specific platforms greatly increases the costs of developing mobile applications to be accessible in a variety of devices. In order to reduce these costs, various frameworks have been proposed in order to produce cross-platform applications.

A. Frameworks

The DIMAG Framework [2] was developed for automatic multiple mobile platform application generation. This was accomplished by creating a declarative definition language which is comprised of 3 distinct parts; firstly a language
DIMAG-root, which provides references to the definitions for workflow and user interface in the application; secondly the language State Chart eXtensible Markup Language (SCXML) defines the workflow by the definitions of states, state transitions, and condition based actions; and finally DIMAG-UI language based on MyMobileWeb’s IDEAL language using CSS to control the user interface. The main shortcomings of this method is that it relies on server-side code generation and download. The other difference with our work is that applications developed in this framework are interpreted using a virtual machine.

Other frameworks include XMLVM [3], [4] developed at San Francisco State University and created to support byte-code cross-compilation and avoid source-code translation through the use of a tool chain. This tool chain currently translates Java Class files and .Net executable to XML documents, which then can be output to Java byte code/.NET CIL or to JavaScript and Objective-C. This tool chain was firstly used to cross compile Java applications to AJAX applications [5], because of the lack of IDE support and difficulty in creating an AJAX application. Further work to include Android to iPhone application cross-compilation [6] was completed. API mapping between the two platforms was carried by the creation of a compatibility library.

More recently, since the arrival of HTML5 and WebKit, a number of open source and commercial cross-platform frameworks have been proposed such as the Appceleratorator (http://developer.appcelerator.com), PhoneGap (http://www.phonegap.com) and Rhomobile (http://rhomobile.com), Frameworks, which use either JavaScript or Ruby, and therefore the resulting applications are run in a browser. Furthermore these applications can run offline and access the device’s full capabilities; such as a GPS or camera; providing the same look and feel as a native application. Although these frameworks greatly simplify the task of implementing the mobile application, the developer is still required to work with general web application languages which lack specificity and efficiency in terms of mobile applications. Furthermore, these applications suffer limited visibility on the market due to the absence of an “official” distribution channel.

There are several different software platforms for mobile applications. Since commercial developers usually wish to develop an application that can work over all platforms there are a number of proposals for single technologies that can target multiple platforms. A recent proposal for a DSL for mobile applications [7] uses XText and Eclipse to implement a DSL that uses code generation techniques to target mobile platforms. This DSL uses fixed GUI structures such as section whereas our language leaves the collection of external widgets as a parameter of any use of the system. It is also not clear whether the DSL has a static type system and its semantics is not defined independently of a translation to a target platform.

Mobl (http://www.mobl-lang.org/) is a language that has been designed to support mobile application development and which targets JavaScript. It has many things in common with our language, however the language features for describing GUI components is fixed and the semantics is not defined independently of the target language.

In all cases the currently available DSLs for mobile applications differ from the language presented in this article in that they are not higher-order. By providing functions as first-class data values, our language provides unrestricted abstraction over mobile programs. Therefore our language can express patterns, such as required by product lines, by parameterising over reusable application elements. Existing languages are also fixed in terms of the external widgets. Our language is parameterised with respect to external widgets and they are simply provided at the type-checking phase when they are checked in terms of the events that they can raise.

Our language is textual and does not aim to provide a graphical DSL that can be used to shield developers from the technical details of application design (although it is intended to shield application developers from the details of mobile platforms). Another approach to DSL design involves graphical modelling languages such as that defined in [8], however it is not clear that such approaches can abstract from technical details without resorting to a textual DSL at some level.

Links [9] is a language that has been designed to support web application development where the 3-tier architecture is supported by a single technology. Like the technology described here, Links supports higher-order functions and is statically typed with respect to events and messages. Unlike our language, Links has been designed as a complete language with supporting tools, and indicates a possible future direction for layering a user language on the calculus.

B. Domain Specific Languages and Modelling

Domain-specific languages (DSLs) have provided the support for software development process by raising the level of abstraction and introducing specialised viewpoints of a certain problem space. The benefit of DSL to application development has been described in [10], [11], [12], [13]. More recent DSLs in other areas include [14] that concentrates on the abstraction of web applications to lower the overall complexity of the application and boilerplate code. Further work on this DSL led to the creation of Platform Independent Language (PIL) [15]. PIL was developed as an intermediate language, to provide a scalable method for developing for multiple platforms. A drawback of this method is currently it lacks support for mobile platform development.

Other efforts for making mobile application development easier include Google Simple (http://code.google.com/p/simple/), a BASIC dialect for creating Android applications, and more recently the Google App Inventor (http://appinventor.googlelabs.com/about/), which is based on Openblocks [16] and Kawa (http://www.gnu.org/software/kawa/). Particularly Google App Inventor has vastly abstracted app development, but only supports development of Android applications.

DSLs for mobile application have been limited due to a number of factors, such as the rapid change of the devices, the closeness of the distribution channel. However, with the
popularity of the native-look-a-like web applications more
DSLs are starting to be developed. An earlier work in this area,
Balagtas-Fernandez have looked at the design of graphical
DSL for non-technical users [17]. This work is still at an
early stage and the tool to support the DSL is a prototype.
However, the initial survey of potential users has shown that
non-technical users would be interested in developing their
own application and would require an easy interface to do so.

More recently, Brenhs has proposed MDSD, a DSL for
iPhone. The language is more specific to data centric ap-
plications. Following from that work, they have started the
Applause project for developing DSL for iPhone, iPad and
Android (http://code.google.com/p/applause/), but this is still
not fully developed.

The SERG group defined a language named Mobl
(http://www.mobl-lang.org/) with a declarative DSL for both
the user interface and for defining the data. Although mobl is
comparable to our language there are substantial differences
since we have aimed to capture the essential features of mobile
applications through the use of: technology independence;
static typing for all features; higher-order functions; formally
defined operational semantics; widget libraries.

Our aim is for the language presented in this article to be a
formal foundation for each of the current implementations of
mobile applications DSLs.

III. Multi-Platform Development and
Contribution

Because of the complexities in multiplatform development,
in this section we introduce three methods that could be used
to help application development for multiple platforms. Our
proposal is to select an approach based on Domain Specific
Languages.

A. Approaches to Development

Frameworks: The use of frameworks can be seen as a
method of software abstraction using common code, which
can be overridden and extended by a user. Within mobile
development, frameworks have been developed to help with
specific tasks including media playback, access to sensors
and graphic and UI manipulation. For example, the system
described in [6] has been designed to help make code bind-
ings between the different platform specific frameworks. This
method concentrates on solving all computational problems,
which can increase complexity in application development,
further becoming a hindrance to the developer.

Web Applications: A mobile web application essentially is a
regular Internet application designed to fit the average screen
sizes of most mobile devices, bringing various benefits to the
developer. Some applications that require high amounts of
processing can greatly benefit from allowing the processing to
be handled in the cloud while the device merely has to process
the UI. In addition the use of standards such as HTML and
CSS may make certain types of applications easier to develop.

Originally this approach was troublesome because of re-
liance on network connectivity for the application, which in
some situations may either not be available or not desired. Web
applications also couldn’t store local data to the web browser,
until the development of HTML5 [18]. In May 2007, Google
released a plug-in for the Firefox web browser, Google Gears
(http://gears.google.com/). This plug-in supports caching of
web applications to allow offline use, and also the ability for a
web application to store data in a local database. This idea has
been integrated into the development of HTML5, a step in the
right direction but there are still remaining issues. Firstly, web
applications in general can have shortcomings in the amount of
rich UI widgets, with animation for certain widget interaction
being increasing difficult to implement in a mobile web
application. Other problems are limitations of the web-browser
on the mobile devices, possibly leading to inconsistencies in
application functionality between different platforms because
of lack of API for using different device components (e.g.
accelerometers, vibration motors, GPS etc). Because of these
limitations and current unsolvable dependencies, the creation
of a Domain Specific Language was chosen as our solution.

Domain Specific Languages: A Domain Specific Language
(DSL) [19], [20] is primarily designed to be used in a
certain application domain (e.g. mobile, telecoms, finance),
abstracting away from the software implementation making
implementation easier. The abstraction is designed to aid
the developer and is to be contrasted with General Purpose
Languages (GPLs) whose features are not designed with any
particular domain in mind. DSLs have existed for many years.
Languages that were created for particular domains include
FORTRAN [21] used to allow direct mathematical formula,
Structured Query Language (SQL) [22] for database access
and manipulation, and Algol [23] for algorithm specification.
In recent times, the use of DSLs have been proposed and used
in different domains including the production of rich web ap-
lications [14], mashups of web applications and services [24],
and system integration [25]. Because of the complexities in
mobile development, we believe there is room for abstraction
in the development for mobile devices.

B. Problem, Proposal and Contribution

There are many different technologies for mobile-
application development. Most of these are complex and do
not separate out the application logic from the GUI imple-
mentation. Such a separation is sensible because the application
logic can usually be reused across multiple platforms while
the GUI implementation must be changed. Furthermore, most
technologies for mobile-application development are event
driven, but dynamically match event-handlers with the events
when they are raised. Our claim is that the quality of devel-
opment can be enhanced by statically checking that handlers
for all events are defined before the application is executed.

Our proposal is that the essential features of languages
for mobile application development have some characteristic
elements that can be captured in a domain specific language.
Furthermore, the language can be constructed as an extension
of the simply typed \(\lambda\)-calculus where the extensions are both
orthogonal and which characterise the domain. Since our
language is based on the \(\lambda\)-calculus it is highly expressive
in terms of being able to parameterise over the elements of an
application and thereby encode patterns of data (e.g. reusable widgets) and control (e.g. call-backs and event handlers).

We use an approach based on monads to contain those parts of an application that deal with updating state (SQLite for example). As described in [26] this supports the desirable situation where applications can be built from composable units. The language has a formally defined semantics that is independent of any implementation technology and therefore can be used as a blueprint for an implementation in any target platform, or can be implemented directly as an interpreter.

The language uses external widgets to define platform-specific features such as GUI elements and persistent storage. The external widgets are fully integrated with the type system of the language and therefore, by supplying different collections of external widgets, the language represents a family of related development platforms. Finally, the language has a type system that allows mobile applications to be statically checked. In particular the events raised by GUI widgets and the device can be checked against handlers defined in the application before it is executed.

IV. DOMAIN ANALYSIS

A DSL is defined by performing a domain analysis [20] on a target family of applications in order to identify the common characteristic features. The domain analysis leads to the design of a technology that conveniently supports these features. Our aim is to define a language that can be used to represent mobile applications and therefore our domain analysis starts with the construction and analysis of two phone applications developed as part of University business and enterprise activities. This section describes the case studies and then outlines the characteristic features that were identified.

A. Case Studies

Two iPhone application case studies were created for a local Small to Medium sized Enterprise (SME). These applications are described in the following two sub-sections.

Tour de France (TDF2009): This application was created to provide access to information from the 2009 series Tour de France cycle race. Firstly the application required a method of transferring and receiving data from an external server for two different reasons. Firstly for the stage results, and secondly for the general data including information about the Teams/Riders and all the Stages involved in that year, this helped us achieve a very small installation size. The data communications were done via XML files parsed using the iPhone SAX-XML Parser, one created with the static data, and one generated every day with the current results. Inside the stages section, fly-through videos to help illustrate the course and terrain, large high resolution gesture controlled pictures were incorporated. Key features of the TDF2009 are shown in Figure 1 where a main screen provides access to the results of current stages and to fly-through videos. The figure shows that the application is driven by events caused by the user touching the screen and that the application consists of different displays (or states) consisting of a mixture of event processors (buttons) and simple information (images, text).

Lyrical Genius: This application was created as a game, Lyrical Genius (LG), consisting of quiz questions relating to different lyrics in songs. This game though still using the Apple Cocoa Framework is quite different to TDF2009 in many ways. Firstly this application does not use XML files as persistence of data, but uses a SQLite database for storing level and question data. Other features of this game include music that is played in the background that can be switched on/off and sound effects for if the user chooses the correct or incorrect answer. These features require threading, which is one issue we must consider in the DSL. The game also includes a timer, for which the user must get a number of correct answers within a time limit. This makes use of threading again, and also another important area as the use of timing can be needed in many different contexts. Features of the application are shown in Figure 2 where a player starts at the main screen. On choosing to play, the user is shown their current score (accessed via the database), if they choose to play then they are offered a difficulty level for each question. Each question has a time-limit; the final screen is reached by either selecting the correct lyric or when the time-out occurs.

B. Domain Features

Based on the case studies above, we can define a set of features that the DSL must support. In the case of GUI implementation, in the iPhone and Android development Openly can be used. Openly is a cross-platform graphics language which supports the ability to draw 2D and 3D objects, but in this paper we are concentrating on the platform framework for the GUI.

Screen Size: Mobiles support only a limited size display. This size leads to a relatively small number of GUI features, therefore there is more scope for building these features into a common language. The standard iPhone resolution is 480 by 320 pixel and the IPA supports a 1024 by 768 resolution. This compares to the Android screens, which vary by hardware vendor but resolutions range to about 480 by 800 pixel. Apple have currently settled the differences in screen display resolution by the use of graphic scaling. This method can seem an effective way of allow iPhone apps to run on a IPA, but this comes with its flaws. Graphic scaling of very small low quality images can make them look unappealing to user. Also differences in screen display resolution can be handled in the iPhone and Android development Openly can be used. Openly is a cross-platform graphics language which supports the ability to draw 2D and 3D objects, but in this paper we are concentrating on the platform framework for the GUI.

Layout Control: Layout control is an important consideration. The iPhone controls layout through the use of XML files, supporting different layout styles. The main style types consisted of linear, relative and absolute. Android now has now deprecated absolute positioning, due to the fragmentation in different hardware vendor screen resolutions (see above). This compares to iPhone, which can do programmatic layout and XML type interfaces using Interface Builder. Interface Builder can help the user easily create UIs, but these layouts would be less
dynamic than programmatic ones. Like screen size, the DSL should focus on application logic and factor out the layout control details into external libraries.

**GUI Element Containership:** Both iPhone and Google Android platforms use a form of GUI element containership. In iPhone development, the emphasis is on the application Window and its Views, withSubviews. These are then 'stacked' onto each other to create anything from a simple to complex interface. With the Android a similar model is used, except with Views and ViewGroups. Interface control on both platform have similarities and differences. On the iPhone, views are normally controlled by the use of View Controllers, which are where widget event handlers are implemented. In comparison Android development uses Intents and Activities. This feature leads us to conclude that all mobile application GUIs can be expressed in terms of a tree of widgets that manage data and behaviour and whose detailed layout and rendering properties can be factored into platform specific libraries.

**Event Driven Applications:** The applications we are targeting are event driven. Most mobile application implementation languages register event handlers dynamically. This method means there is a lack of checking at compile time to prevent an application crashing. An example of a event listener for iPhone:

```plaintext
[btnMenu addTarget:self action:@selector(backToMenu) forControlEvents:UIControlEventTouchUpInside];
```

If the action is not registered then the program might go wrong. This is an issue in general with event driven programming and in particular with mobile applications where events can be supplied by both the platform and the user. Contextual events such as platform orientation, GPS, and battery levels should be handled by an application in suitable ways. This places a desirable feature requirement on our DSL whereby the presence or otherwise of event handlers can be detected at compile-time.

**Hardware Features:** Modern day mobile devices come equipped with many different features. These features include microphones, accelerometers, GPS, camera, and close range sensors. These features tend to be fairly standard in their behaviour if they are supported by the platform. Although many platforms have comparable hardware features, they differ in the details of how to control and respond to them. The DSL should allow the details of hardware to be factored out into platform specific libraries whilst supporting the events and controls associated with them.

**Concurrency:** The use of concurrency in mobile applications is paramount. This is carried out by the use of threads, for instance a UI thread starts with the execution of an iPhone or Android app. Because this thread is used for the UI elements of the application, heavy or concurrent tasks should be allocated in its own thread. This can help avoid UI halts and a 'laggy' experience for the user. On the iPhone platform, threads can
be implemented in various ways including POSIX Threads and NSThread. The difference between the two are that the pThreads are a C/C++ library and NSThread is a Cocoa-native thread. On Android, concurrency can be implemented through the use of Thread Classes, just as you would do it Java. Example of a thread in iPhone:

```c
[NSThread
 detachNewThreadSelector: @selector(playMusic)
toTarget:self withObject:nil];
```

Although threads are important, we are proposing a DSL for mobile information systems rather than applications with real-time features such as games. Therefore, it is not clear that very fine-grain control over threads will be important. Rather, it is likely that lightweight concurrent processes are require where control is fairly simply in terms of thread interruption and resumption. Furthermore, we will assume that threads can be associated with components of an application and their control can be integrated with application events. Therefore, the DSL will support lightweight threads, but not necessarily provide any special purpose features for creating and manipulating them.

**Object-Orientation:** Mobile applications are typically Object-Oriented (OO). In the iPhone the main language used is Objective-C, though support for C++ and the non-OO C can also be used. This compares to Android, which uses Java, but with different libraries and uses the Dalvik Virtual Machine (VM) instead of the Java VM, because its characteristics support mobile devices more. Applications are built by constructing new and extending existing class/object types. The DSL should have OO features including the ability to encapsulate state and associate methods with GUI widgets.

**Transitional Behaviour:** State machine transitional behaviour is very common in mobile device applications, and can be found on the Android platform. Each Activity can be viewed as a state machine that stores state and actions by the user, which then causes transitions between different views or activities. The DSL will need to support a state-machine view of mobile applications.

**Data Persistence:** Mobile applications and increasingly persisting data to physical storage between application invocation. This data can sometimes be as simple as general settings that the application needs, but also can also be quite complex and including high amounts of redundancy. Modern smartphone platforms currently have implementations of a SQLite (as in LG above), a lightweight serverless single file database engine. Other methods of storage can be in the forms of general binary/object files that store serialisable objects and which is not highly portable, and XML (as in TDF2009 above); highly portable but requires more storage space, and also can require large amounts of parsing which is not ideal. Therefore, whilst mobile applications require data persistence, the format of the persistence differs between platforms. The DSL must support data persistence and allow the complete state of an application to be saved between invocations; however the details of the data format should be factored out into application-specific libraries.

**Contextual Events:** Within a mobile application, not all events are directly invoked by the user. Mobile platforms have to deal with event invocation from a range of different sources based on its current contextual environment. For example, when the battery is lower on a phone normally the phone will display a message to the user to recharge the battery.

**Static Typing:** type systems are used in programming
languages as a method of controlling legal and illegal program behaviour. Static typing differs to dynamic typing in many ways. Firstly, static typing requires all type checking to be carried out during run-time, as opposed to dynamic typing that requires checking at run-time. Static typing requires explicit declaration of types unlike dynamic typing shown below:

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>// static declaration</td>
<td>// dynamic declaration</td>
</tr>
<tr>
<td>// of integer foobar</td>
<td>// and use of foobar</td>
</tr>
<tr>
<td>int foobar;</td>
<td>foobar=10;</td>
</tr>
</tbody>
</table>

Though the use of dynamic typing may look attractive as variable initialisation is not required, issues can arise from mistyped variable names. With static typing, if a mistyped variable name is used within a method, the source will not compile, whereas using dynamic typing it will compile making software debugging a much harder task. Therefore the DSL should have a type system that allows as many errors to be caught at compilation-time as possible.

C. The Mobile Application Domain: Conclusion

This section has reviewed two mobile applications and has identified features that are common to the application domain. We have identified the domain as including mobile information systems: the family of applications that process information, are event driven, use contextual information from the platform, have relatively simple hierarchical GUIs, and use simple concurrency. This excludes applications with sophisticated GUIs or that require complex real-time processing, for example games. The proposed DSL should exhibit these characteristic features without unnecessary implementation considerations, factoring them out into libraries, so that it can form the basis of analysis and implementation of more practical languages for mobile applications. The rest of this article describes the DSL, illustrates its use with some examples, and analyses its features.

V. MOBILE DSL

A. Overview

Although each mobile platform has a different implementation language, the key features are the same. Our hypothesis is that we can capture the characteristic features in a single language that forms a basis for all other mobile application languages. In order to do so our language must be highly expressive and contain features that support those outlined above. A standard starting point for such a language design is the \(\lambda\)-calculus which is highly expressive, flexible, supports analysis, and is readily extended.

Our language is an extension of a \(\lambda\)-calculus where characteristic features described above are supported as follows:

**Widgets:** Our language provides a special widget-definition feature. A widget consists of state, behaviour and event handlers. The details of how a widget is rendered and how it runs on the particular mobile platform is outside the scope of our language, however the application logic for each widget is completely defined in the language.

**External Widgets:** The language is parameterized with respect to the basic widget library that is used in any given application. This allows issues such as screen size, layout, hardware features and concurrency to be separated from the application logic. The external widgets are characterized by their type signatures that allows the application to be statically checked.

**Containment:** Our language uses a simple notion of containment via widgets, records and lists to express the structure of a mobile application GUI. Furthermore, the semantics of our language (described below) uses the widget containment tree as the basis for handling events.

**Events:** The semantics of the language is partially defined in terms of handling a sequence of externally generated events. An event occurs at a particular widget, for example the user presses a button. If the widget defines a handler for the event then the event can be processed. If the widget does not define a handler then the search proceeds up the widget containment tree until a suitable handler is found. Static type checking guarantees that an event handler will be found. An event handler is responsible for performing any state updates and then returning a replacement widget for the receiver of the event.

**State:** Each widget has local state that can be updated by performing commands. The extent of the local state is the lifetime of the application. State that has wider extent than the life-time of the application is accommodated by providing the application with predefined variables, for example a database that is shared between applications.

**Types:** The language has a type system defined in terms of a relation that associates a type to each program. Our claim is that we can construct a static type checker for this type system.

Since the language is an extended \(\lambda\)-calculus, it is very expressive (although expressivity is restricted via static typing). The extensions defined above are novel and allow characterise features of mobile applications to be conveniently expressed. Another novel feature of the language is the semantics that follows a cycle, given a program \(prog\) and state that contains update-able memory locations:

```plaintext
screen := empty;
while true {
    command := reduce(prog);
    (widget, state) := perform(command, state);
    screen := replace(screen, widget);
    event := wait_for_event();
    prog := handle(screen, event);
}
```

The program \(prog\) is reduced (evaluated) to produce a command. A command must be performed with respect to a current state (the database) resulting in an updated state and a widget. The first time the loop is performed, the widget must be a home screen, otherwise \(widget\) is a replacement for the receiver of the most recently processed event. The mobile platform then waits until it received an \(event\). The event is delivered to the widget that defined an appropriate handler for it returning the body of the handler which is a new program.
B. Syntax

This section defines the syntax of our language. The complete language that can be used to write mobile applications is defined in V-B1. The complete language is an expression language that evaluates to produce a value; the language of values is defined in section V-B2. Some values are commands as defined in section V-B3. Together, program expressions, values and commands form the basis of the evaluation cycle defined in the previous section. The distinction between program expressions, values and commands is defined in section V-B2. Some values are commands rather than tokens.

- figures

1) General Syntax: The syntax of mobile application programs is shown in Figure 3. The features of the language are divided into: basic λ-calculus; data extensions; commands; widgets. The basic λ-calculus consists of variables, constants, functions and applications. Simple extensions to the calculus include conditionals and local variables. A mobile application needs to represent data structures and these are expressed in terms of lists and records. For convenience we allow definitions in let, records, and widget to be written f (x, y, t) instead of f = fun (x, y) t.

Commands are used to access and store values in memory locations. A command can allocate a new memory location and initialise it, can access the value of a memory location and can set it. A command sequence is a special type of term that is used to place an ordering on the evaluation of commands (execution as defined below is otherwise unordered). Suppose that we want to create a function that allocates a two dimensional point and provides an operation to move it:

```
let x = loc1 in
let y = loc2 in
let loc2 = loc(0) in
mv = fun (dx, dy) {
    xval <=> get(loc1),
    yval <=> get(loc2),
    xval += dx,
    yval += dy
} |
widget(t; t) = t
```

When the function is called it returns a record with three fields: x, y and mv. The record is allocated by a command sequence in lines 2-9. A command sequence has the form { t | bs } where t is a term that constructs the return value of the command sequence and bs is a sequence of commands. Each command is performed in turn and produces a value that is bound to a variable, the commands on lines 10 and 11 allocate new memory locations, initialised to 0, and then binds them to the variables loc1 and loc2. The variables are scoped over the value returned by the command sequence (and mutually recursively over each other).

The record returned by the function is defined on lines 2-9. The record has three fields. The first two fields on lines 2 and 3 just associate the field names x and y with the allocated memory locations. The third field called mv is defined on lines 4-9. The move function takes two arguments dx and dy and will move the allocated point by updating the memory locations. To do this it must access the current contents of the locations, add in the deltas and then update the locations. This is done using a nested command sequence in lines 5-9.

The nested command sequence returns void on line 5. The value of void is bound by the command sequence but is not important since the commands are performed in order to update the locations, not for their return value. The command sequence accesses the current memory contents (lines 6 and 7) and then updates them (lines 8 and 9). Note that these are performed in sequence.

Finally, terms may be widgets. A widget has the form: widget(ext, id) state handlers where ext is an external widget reference, id is an identifier for the widget, state is a record of state variables for the widget and handlers is a record of functions that implement event handlers for the widget. The external widget is a reference to a library element that determines how to display the widget on the mobile GUI. The external widget places requirements on the state of the widget and handlers that must be implemented. A widget is free to define extra state and handler elements than those defined by its external widget.

For a widget to be correctly formed, each of the handlers must be a function that returns a command. Furthermore, when the command is evaluated it must return a widget. This ensures
that the execution cycle for the language works correctly. These formation rules are enforced by the type system as described below. Finally, a program is correctly formed when it is a command that returns a widget whose external widget is a screen.

Figure 5 shows an example application that implements a simple contacts database. The application has two top-level states: main and done. The application starts in state main and displays a text input field and a button. Each time the text changes in the input field, a textChanged event is received. If the button store is pushed, then the current database of contacts (labelled contacts) is updated and the machine makes a transition to the state labelled done. When in the done-state, the application displays the current list of contacts and displays a button labelled ok. When the ok button is pushed, the application transfers back to the main screen to allow more contacts to be entered.

The program that implements this application is shown in Figure 4. The program consists of a single command sequence that returns the main screen. Each top-level command defines a widget. For example, the main widget (lines 2 - 12) is based on the external widget called Screen and with widget id 0.

The main screen widget has two state variables called contacts and contents. The contents variable must be present because the external Screen widget requires the contents to be defined as a single sub-widget (in this case a table on line 13). The contacts variable is used to store a list of contacts where each contact is a string. Since contacts will be updated, it must be a memory address that is allocated by a command which is shown in line 11.

The main screen widget defined a single handler called push (lines 5 - 10). A handler is a function. Events occur on external widgets and the owning widget is checked to see if it defines a handler with the appropriate name. In this case the main screen contains a sub-widget called store (line 14) that can process a push event due to its external widget being of type Button. Since the widget does not implement any handlers, the event will be promoted to the most immediately containing parent widget, in this case the table. Event promotion continues until an appropriate handler is found, in this case it will be the definition of push in the main screen.

Similar processing occurs when the text is changed in the text input field defined in lines 15 - 20. In this case the Text external widget generates a textChanged event that is handled locally by the owning widget. When an event is handled, the associated function is called, supplying it with any associated arguments. In the case of changing the text, the textChanged handler receives the text in the field as shown on the screen. The body of a handler must be a command that returns a widget. The commands are performed and the returned widget becomes a replacement for the widget that handled the event. In the case of textChanged the body of the handler updates the contents of the string location. In the case of push in the main widget, the command sequence retrieves the current list of contacts (line 7), gets the contents of the text field (line 8), updates the current contacts database
(line 9) and calls the function done (line 6). The function
clone maps a list of contacts cs to a widget that displays the
contact strings in a text list and provides a button that returns
to the main screen. Like the main widget, the push event
generated by the button on the done screen is handled by
a function defined by the top-level widget. The body of the
handler returns the main screen, therefore, pushing the OK
button returns to that screen as required.

The code above is implemented by the following Java class:

```java
class Cell {
    Object c;
    Object v = c;
    c = n;
    return v
}
```

### C. Types

<table>
<thead>
<tr>
<th>α, β ::=</th>
<th>types</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>type variables</td>
</tr>
<tr>
<td>λX.α</td>
<td>type abstractions</td>
</tr>
<tr>
<td>α[α]</td>
<td>type instantiations</td>
</tr>
<tr>
<td>α + β</td>
<td>type alternatives</td>
</tr>
<tr>
<td>bool, int, str</td>
<td>type constants</td>
</tr>
<tr>
<td>α[0,n] → a</td>
<td>function types</td>
</tr>
<tr>
<td>[0]</td>
<td>list types</td>
</tr>
<tr>
<td>{x_i : α_i[0,n]}</td>
<td>record types</td>
</tr>
<tr>
<td>loc(α)</td>
<td>location types</td>
</tr>
<tr>
<td>state → (α, state)</td>
<td>command types</td>
</tr>
<tr>
<td>ω(α, α) {x_i : α_i[0,n]} {x_i : α_i[0,m]}</td>
<td>widget types</td>
</tr>
<tr>
<td>ξ{x_i : α_i[0,n]} {x_i : α_i[0,m]}</td>
<td>external types</td>
</tr>
</tbody>
</table>

Evaluation of programs relies on the following properties:

- A mobile program is a command that processes a state
  which is a local database for the application. A program
  is therefore a function from states to states.
- A mobile program must define the correct state compo-
  nents for the widgets it displays. For example, a button
  must have a label and a text field must be associated with
  a memory address that can be updated with changes to
  the text as it is typed.
- The GUI of a mobile application is a hierarchically or-
  ganized tree of widgets. Not all combinations of widgets
  are legal, for example a table may not contain screens.
- A GUI program must return a widget of type Screen
  at the top-level.
- A mobile program must define handlers for all the events
  that can occur when the user interacts with the GUI
  or when the underlying mobile device changes state
  (orientation, battery charge levels, GPS, etc).
- A mobile application must specify a state transition
  when an event occurs. The transition must specify a
  replacement for the widget that receives the event.
Each of the properties listed above should be checked before the program is executed. Most mainstream mobile application languages only support checking a sub-set of these properties. For example, event handlers are usually registered dynamically which means that not all events may have an appropriate handler defined at run-time.

Our language has a static type system that checks all of the properties listed above. All values have types represented using the type language in Figure 8.

Type variables, type abstraction, type instantiation and type alternatives are used to implement parametric polymorphism to allow, for example, functions over lists of several value types. In particular we need to be able to define widgets, for example Table, that are generic with respect to their contents.

Type constants, function, list and record types are standard. The memory address containing a value of type $\alpha$ is of type $\text{loc}(\alpha)$. A command must process the application's local database. The type of a command is $\text{state} \rightarrow (\alpha, \text{state})$ where $\alpha$ is the type of the value returned by the command. The type is intended to imply a function from states to states such that command lists must be ordered by combining functions. This construction is exactly how monads are encoded in functional programming languages.

Widgets have a type that encodes the type of their external widget, the type of their identifier, the type of their state variables and the type of their handlers. An external widget has a type that defines the requirements on the state, methods and the events that are produced.

A type relation for the language is defined in Figure 9. The relation has the form: $\Gamma \vdash t : \alpha$ where $\Gamma$ is a type judgement mapping variables to types, $t$ is a term and $\alpha$ is an associated type. A command is a term with the following type:

\[ \Gamma \vdash \text{command} : (\alpha, \text{state}) \]

and a program is a command where $\alpha$ is the following type:

\[ \omega(\text{Screen}, \text{int})\{\text{contents} = \beta, \ldots\}\{\ldots\} \]

for some appropriate widget type $\beta$ and associated handlers. Since the types of generated events are encoded in the external widget types and the types of handlers are statically determined in a program, it is possible to statically check that all events have an appropriate handler, i.e. that no event will be lost.

\[ \begin{align*}
\text{events}(\alpha + \beta) &= \text{events}(\alpha) \cup \text{events}(\beta) \\
\text{events}(\text{bool}) &= \emptyset \\
\text{events}(\alpha_i : [0,n]) &\rightarrow \alpha_i = \emptyset \\
\text{events}([\alpha]) &\rightarrow \text{events}(\alpha) \\
\text{events}(\{x_i : \alpha_i\}) &= \bigcup_{i \in [0,n]} \text{events}(\alpha_i) \\
\text{events}(\text{loc}(\alpha)) &\rightarrow \text{events}(\alpha) \\
\text{events}(\text{state} \rightarrow (\alpha, \text{state})) &= \emptyset \\
\text{events}(\omega(\alpha, \_)) \rightarrow \text{events}(\alpha) \\
\text{events}(\{x_i : \alpha_i\}) &\rightarrow \{x_i : \alpha_i\} \\
\text{events}(\xi(\_)) &\rightarrow \{x_i : \alpha_i\} \\
\end{align*} \]

Figure 10. Widget events

D. Events

An important feature of many programming languages is the ability to catch errors as early as possible. Programming language types are used to prevent incorrect data being supplied to operations. Languages with dynamic typing leave type checking to run-time whereas static typing allows a program to be checked before it is executed.

Our language has a type system as described in 9 that is intended to be statically checked. Although no type checker is presented here, the type relation is relatively standard and therefore we claim that a static type checker for the language is straightforward.

In addition, the language has been designed to allow events that can be raised by widgets to be statically matched against handlers. This feature is unusual amongst languages that support event driven GUIs and this section describes how the checking is performed.

A mobile program is a command that returns a widget of the following type: $\omega(\text{Screen}, \text{int})\{\text{contents} = \beta, \ldots\}\{\ldots\}$ for some widget type $\beta$. This type represents a tree of widgets rooted at a screen. The sub-trees and leaves of the tree are widgets that can raise events when the user interacts with them or when the state of the underlying mobile platform changes. The semantics of the language allows the handlers for events to be defined by either the widget that raises the event or a containing parent widget. Therefore, to check that handlers are defined for each event that can be raised by a program, it is necessary to construct the set of outstanding events for each widget in the tree. This is defined in Figure 10 such that a program $p\alpha$ is well-formed when $\text{events}(\alpha) = \emptyset$.

VI. Execution

The execution of terms in the language occurs on a hypothetical virtual machine whose states consist of terms, memory states, and a sequence of events. The machine executes by performing a sequence of steps that reduce the term with respect to the memory and the input events. The execution is performed in a particular order so that the expressive properties of the higher-order language are preserved even though an application performs side-effects with respect to the state and event stream. Preservation is important in order that the mobile applications can take advantage of higher-order features including parameterisation over all language features (for patterns, product lines etc) and first class functions (continuations, control abstractions). Execution occurs in the following sequential phases:

1) reduction of a term to a command.
2) performing a command with respect to a state to produce a widget and an updated state.
3) replacing the receiver of the most recent event with the widget.
4) handling the event by calling a function that produces a new term.

The first time the sequence is performed, there is no receiver therefore step 3 produces a screen. On subsequent iterations, the term produced in step 4 is used in step 1. The rest of this section describes each of the phases in turn.
Figure 9. The type system for the mobile application language

### A. Term Reduction

Figure 11 shows the definition of an evaluation relation that reduces a term to a value or normal form. The relation is a small-step semantics for the language meaning that its reflexive, transitive closure $\rightarrow^*$ defines an execution trace for any given term. Notice that the relation does not impose any unnecessary execution ordering on a composite term.

### B. Performing Commands

A command is a particular type of term that acts as a function from states to values and states. A state is a mapping from memory addresses to values and a command may access the contents of an address, update the contents of an address or both. In all cases a command returns a value, but in some cases the value is irrelevant because the command is being used for its side effect on the state. Figure 12 defines a relation: $\Sigma \vdash c \Rightarrow v, \Sigma'$ where $\Sigma$ is a state before the command $c$ is performed to produce the value $v$ and the resulting state $\Sigma'$. There are three atomic commands and a command sequence. The atomic commands are: get which accesses a memory location but does not change the state, loc which allocates a new memory location and returns it; set which updates a memory location (and returns the old value which is usually ignored).

A command sequence has the form $\{ t \mid x_1 \leftarrow c_1, x_2 \leftarrow c_2, \ldots, x_n \leftarrow c_n \}$ where each command from $c_1$ down to $c_1$ is performed in turn. The results of the commands are bound to the associated variables in parallel which means that the bindings are mutually recursive whilst command execution is in sequence. The result of the command sequence is the value produced by the body $t$ in the context.

<table>
<thead>
<tr>
<th>T-VAR</th>
<th>$\Gamma \vdash x : \alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-TRUE</td>
<td>$\Gamma \vdash \text{true} : \text{bool}$</td>
</tr>
<tr>
<td>T-FUN</td>
<td>$\Gamma \vdash \text{fun}(x_1 : \alpha_1, \ldots, x_n : \alpha_n) \rightarrow \alpha$</td>
</tr>
<tr>
<td>T-IF</td>
<td>$\Gamma \vdash \text{if} \ t \ \text{then} \ t_1 \ \text{else} \ t_2 : \alpha$</td>
</tr>
<tr>
<td>T-REC</td>
<td>$\Gamma \vdash { x_i : \alpha_i }<em>{i \in [0, n]} : { x_i : \alpha_i }</em>{i \in [0, n]}$</td>
</tr>
<tr>
<td>T-LIST</td>
<td>$\Gamma \vdash [t_i : \alpha_i] : \alpha$</td>
</tr>
<tr>
<td>T-LOC</td>
<td>$\Gamma \vdash \text{loc}(t) : \text{state} \rightarrow (\text{loc}(\alpha), \text{state})$</td>
</tr>
<tr>
<td>T-MON</td>
<td>$\Gamma \vdash { t \mid x_i = t_i }_{i \in [0, n]} : \text{state} \rightarrow (\alpha, \text{state})$</td>
</tr>
<tr>
<td>T-WID</td>
<td>$\Gamma \vdash \text{widget}(t_1, t_2) { t_i = u_i }<em>{i \in [0, n]} : \omega({ x_i : \alpha_i }</em>{i \in [0, k]} \rightarrow \alpha, { x_i : \alpha_i }_{i \in [k+1, n]} \rightarrow \beta)$</td>
</tr>
<tr>
<td>T-EXT</td>
<td>$\Gamma \vdash \text{ext}(k) : \xi$</td>
</tr>
<tr>
<td>T-OPT-1</td>
<td>$\Gamma \vdash t : \alpha \ + \ \beta$</td>
</tr>
<tr>
<td>T-OPT-2</td>
<td>$\Gamma \vdash t : \alpha \ + \ \beta$</td>
</tr>
</tbody>
</table>

$\Gamma \vdash t : \lambda X. \alpha$

$\Gamma \vdash t : \alpha | X \rightarrow \beta$
of the variable bindings.

C. Event Handling

An event consists of a name \( n \), a widget identifier \( i \) and some argument values \( \{i \in [0:n]\} \). The event occurs with respect to a widget \( w \) somewhere on the current screen \( w_s \). The most deeply nested enclosing parent \( w' \) of \( w \) (where parent is a reflexive transitive relation) that defines a handler named \( n \) is selected to handle the message:

\[ i, n \vdash w_s \Rightarrow w' \]

where the event handling relation is defined in Figure 13. In order for no event to be lost we need the following proposition to hold:

**Proposition 1.** If \( \Gamma \vdash \alpha \) and \( \text{events}(\alpha) = 0 \) then \( w' \neq \epsilon \).

The handler \( w', n \rightarrow \text{fun}(\alpha) \) is supplied with the argument values and the result of the handler must be a command \( c \) as required by the following type constraint on the program:

**Proposition 2.** If \( w', n \rightarrow \text{fun}(\alpha) \) then \( k \) is a handler then
Therefore we have formally specified mobile applications. It is not intended to directly support be used to analyse domain specific languages that support
to directly support mobile applications in its own right, in that sense it is a mobile application calculus. The calculus is executable, and therefore can be used as an intermediate language as shown in Figure 14 where the architecture consists of 3 tiers: (1) the application, written and compiled using a DSL; (2) the DSL specific engine and appropriate libraries; (3) the running platform, Java, C#, .NET, Android or iOS (iPhone). For each of the target platforms, the engine will comprise of two major parts. Firstly there will be the platform libraries (MobLib) that contain the specific platform API calls. This library will contain the callable display, interface, and underlining methods of the platform. Secondly the engine, that will run the compiled code and make the appropriate platform calls using the the bundled platform library set.

VII. Analysis

A. Use as an Intermediate Language

The language proposed in this article is a tool that can be used to analyse domain specific languages that support mobile applications. It is not intended to directly support mobile applications in its own right, in that sense it is a mobile application calculus. The calculus is executable, and therefore can be used as an intermediate language as shown in Figure 14 where the architecture consists of 3 tiers: (1) the application, written and compiled using a DSL; (2) the DSL specific engine and appropriate libraries; (3) the running platform, Java, C#, .NET, Android or iOS (iPhone). For each of the target platforms, the engine will comprise of two major parts. Firstly there will be the platform libraries (MobLib) that contain the specific platform API calls. This library will contain the callable display, interface, and underlining methods of the platform. Secondly the engine, that will run the compiled code and make the appropriate platform calls using the the bundled platform library set.

VII. Analysis

A. Use as an Intermediate Language

The language proposed in this article is a tool that can be used to analyse domain specific languages that support mobile applications. It is not intended to directly support mobile applications in its own right, in that sense it is a mobile application calculus. The calculus is executable, and therefore can be used as an intermediate language as shown in Figure 14 where the architecture consists of 3 tiers: (1) the application, written and compiled using a DSL; (2) the DSL specific engine and appropriate libraries; (3) the running platform, Java, C#, .NET, Android or iOS (iPhone). For each of the target platforms, the engine will comprise of two major parts. Firstly there will be the platform libraries (MobLib) that contain the specific platform API calls. This library will contain the callable display, interface, and underlining methods of the platform. Secondly the engine, that will run the compiled code and make the appropriate platform calls using the the bundled platform library set.
B. Implementation Options

Attempting to develop for mobile platforms is a challenging task, and in most cases different approaches come with their advantages, and their disadvantages. With the DSL and the creation of virtual machines on targeted platforms, one particular benefit would be the avoidance of application installation source lock-in, which is applicable to the many users of iPhones/iPads. Through lock-in comes increased security for end-users through the use of application validation by that platform vendor; this can be seen as a method to allow that vendor to decide on the types of applications it believes are right for that user/device.

A VM and downloadable programs (in the form of DSL program definitions), this can be overcome after the user has the VM installed on their device. This requires the VM needs to be downloaded onto the mobile platform; a situation that some vendors take steps to prevent. For example, Sun Microsystems attempted to make available an iPhone version of the Java VM, which includes JavaME, a branch of Java designed for mobile and embedded devices; unfortunately this was blocked. If there was the ability to run a VM on an iPhone/iPad, development of these applications will no longer require a certain platform, as the tools will be operable in multiple desktop platforms. Other advantages of a VM would also include the decrease in application size and faster download times.

An alternative approach to mobile applications involves the use of web-browser technologies such as JavaScript, HTML5 and CSS. These are particularly attractive since an application is portable across many different devices and the introduction of new features in these technologies makes it possible to offer many of the features of native applications. The architecture described in Figure 14 can be used with these technologies in order to offer abstraction through a domain specific solution.

C. Current State and Further Work

The calculus language was initially described in [27] and has been prototyped as an interpreter in Java and used to implement a number of simple applications including a simple address book and a mobile platform adjacency notification application. The next step is to develop a mobile Virtual Machine (VM) for platforms including Android and iPhone. Because of the policies in place regarding VM development for the iPhone discussed earlier, the VM may not be accepted by Apple to be on the App-Store. One method of possibly getting around the issues with the App-store policy on VM development, could be incorporating the XVMLVM [3] and instead of following a VM approach, use a compilation approach for iPhone/iPad, or targeting JavaScript.

Because of the problems that can occur from dynamic event and event handler association, we hope to implement a type checking system. In iPhone applications, certain UI classes in the UIKit framework can create events, with the developer then can associate and link with a particular event handler. At compile time, these associations are not checked, and can cause an application to hang and crash if and when that event handler does not exist or meet the requirements of the event. A type checker will be needed to detect situations where event handlers are not implemented. The type checker has been implemented and has been shown to work correctly with a number of example applications. A next step is to provide the consistency and completeness of the type system and to integrate it with software engineering tools such as XText on Eclipse.

Other areas of future work include the ability to connect to external services. This will take the form of a method of connecting to RSS/XML feeds including a method for parsing the documents. Features such as external connectivity can be implemented as external widgets that integrate seamlessly with the language presented here.

REFERENCES


Dean Kramer is currently a PhD student in the Centre for Model Driven Software Engineering at The University of West London, UK where he received his BSc Computing and Information Systems in 2008. His research interests include Software Product Lines, Context-Awareness, Mobile Development, MDA, and Domain Specific Languages. Dean has worked on a number of research projects involving the development of mobile applications and social networking.

Tony Clark is a Professor of Informatics and Head of Department for Business Information Systems at Middlesex University, UK. His research interests include System Modelling and MDA, Domain Specific Languages, Language Oriented Programming, Dynamic Modelling, MetaModelling, Software Tools, Formal Methods, Programming Language Design, and Functional Programming. Prof. Clark has been involved in a number of industrial projects including contributing to the UML 2.0 standard and developing commercial tools for model driven development. He has published widely in the fields of software modelling and programming languages.

Samia Oussena is a Reader and the head of the Centre for Model Driven Software Engineering at The University of West London, UK. She manages a number of funded projects for industry and research bodies. She has a research background in methodologies and software application development. Prior to academia, she gained an extensive industrial experience in software development. Samia research interests are in developing software methods for enterprise applications.
Investigation on Composition Mechanisms for Cyber Physical Systems

Kaiyu Wan, Danny Hughes, Ka Lok Man, Tomas Krilavicius, and Shujun Zou

Abstract—A wide variety of programming abstractions have been developed for cyber-physical systems. These approaches provide support for the composition of cyber-physical systems from generic units of application functionality. This paper surveys the current state-of-the-art in composition mechanisms for cyber physical systems and reviews each approach in terms of its support for composition analysis, re-use and adaptation. We then review approaches for modeling and verifying cyber-physical application compositions and conclude by proposing promising research directions that will address these shortcomings.

Index Terms—Composition mechanisms, programming paradigms, cyber physical systems

I. INTRODUCTION

Cyber-Physical Systems (CPS) are inherently heterogeneous in their components, design requirements and infrastructure [1]. Automated composition is a cost-effective technique of engineering such complex systems. In general, effective component-based system design depends on two properties: compositionality and composability. If these two features are not satisfied, the developed systems will not behave as expected and may be hard to maintain. Hence we argue that the composition mechanism in CPS involves two levels of composition. At one level, composition modeling shall be used to ensure the compositionality of components during design phase. At the second level, application composition approaches shall be used to ensure the composability of systems during developing phase as well as at run-time.

Automated validation and verification of composition modeling holds particular benefits in the field of CPS, where limited resource availability demands that applications operate within strict constraints. While significant work has been performed on analyzing and verifying the composition models [9], [10], this field remains largely theoretical and current verification and validation approaches cannot be used effectively with popular embedded composition mechanisms.

Recent years have seen a proliferation of development approaches for networked embedded systems, each of which can be applied to build Cyber Physical Systems (CPS). These approaches include monolithic paradigms such as embedded C, modular approaches such as Java ME [2] and Contiki [3], static component models such as NesC [4], dynamic component models such as OpenCOM [5] and LooCI [7] and service-oriented approaches such as REMORA [8]. These approaches each offer distinct mechanisms for composing applications from units of generic functionality.

While the research community has provided a rich variety of development paradigms to date little attention has been paid to analyzing the specific trade-offs offered by each approach and discussing how a particular composition mechanism may be matched to the requirements of the final application composition. For example, a monolithic approach provides very little support for reconfiguration but allows for whole-program analysis and verification. Conversely, a dynamic component model may allow for unlimited reconfiguration but at the expense of whole-program analysis and verification. This paper seeks to analyze these trade-offs in detail.

This paper first surveys available composition models and highlights those with particular potential in the field of CPS. The paper then clarifies the specific trade-offs offered by each approach and discussing how a particular composition mechanism may be matched to the requirements of the final application composition. For example, a monolithic approach provides very little support for reconfiguration but allows for whole-program analysis and verification. Conversely, a dynamic component model may allow for unlimited reconfiguration but at the expense of whole-program analysis and verification. This paper seeks to analyze these trade-offs in detail.

The remainder of this paper is structured as follows. Section II discusses the characteristics of Cyber-Physical Systems. Section III identifies key research challenges in composition mechanism. Section IV discusses the some of the most know composition models, which might be the candidates for Cyber-Physical Systems. Section V surveys application composition mechanisms for Networked Embedded Systems, which might be useful for Cyber-Physical Systems. Section VI proposes a methodology for composing and modeling Cyber-Physical Systems applications. Finally, Section VII discusses directions for future work.

II. CYBER PHYSICAL SYSTEMS

Cyber-Physical Systems (CPS) are integrations of computation with physical processes, wherein networked embedded computers monitor and/or control physical processes based upon local (i.e. in-network) and remote (i.e. back end) computational models [11]. CPS tend to feature a tight coupling between physical and software components and may be required to operate for long periods without human intervention. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. In contrast
to traditional embedded systems, CPS interface directly with the physical world. Compared with hybrid systems which deal with physical processes as well, CPS can be considered as a group of hybrid systems which interact and collaborate with each other through networks (including wireless medium). By merging computing and communication with physical processes CPS allows computer systems to monitor and interact with the physical world. The key three “C” conception of Cyber-Physical System are shown in Figure 1. The physical platforms which support CPS offer five capabilities. These are computation, communication, precise control, remote collaboration and autonomy.

Fig. 1. 3C conception of cyber-physical system

CPS tend to feature a tight coupling between physical and software components and may be required to operate for long periods without human intervention. As CPS continuously interact with the physical world, the behavior of a CPS must change in order to maintain optimal operation in the face of changing environmental conditions and operational contexts. In addition, CPS often involve mobile system elements and may be required to operate on battery power alone for extended periods. The Applications of CPS include high confidence medical devices and systems, traffic control and safety, advanced automotive systems, process control, energy conservation, environmental control, avionics, instrumentation, critical infrastructure control, distributed robotics (telepresence, telemedicine), defense systems, manufacturing, and smart structures [11].

In Figure 2 a tour guide example is illustrated. Different tour guide devices (cell phone, PDA and sensors) and tour information systems (Tour information server in this example) will be connected through wired and wireless network to form a secured, reliable and privacy-conserving tour guide service. The system aims to provide tour information accurately, book tickets at run-time, reduce expense effectively and guarantee tour information services exist anytime and everywhere. The description of the system is as follows:

- Tourists can ask for tour information services while driving through cell phone (Tourist A), or sensor (Tourist B) or PDA (Tourist C, D), or by call from home (Tourist E).
- Tourists can get run-time information at a specific scene (Tourist C, D), move around (Tourist B), or call service at home (Tourist E). For tourist A, whenever he makes a phone call, he can get the nearby scene introduction. For tourist C, as long as he is within the space of the scene, he will be able to obtain the tour introduction related to scene 1. However if the introduction is given to tourist C once, no further introduction should be provided unless the tourist C asks for replaying the introduction. For tourist B, if he enters into the space of scene 2, he will be provided the related information.
- All the information is organized obeying certain privacy and security policies in the server. If one scene is too crowded, no further services should be provided (in reality, no tickets can not be sold then). If one tourist has bought too many tickets at the same time, the tourist’s further requirement may be suspended since the person may be suspected as reselling the tickets. Once the tourist decides to go to a scene, the tourist can buy the tickets through their devices.

The above example shows some characteristics of CPS shared with other typical applications of CPS such as health monitoring systems, traffic monitoring and control, process control, energy conservation and environmental monitoring [12]. Many of these systems are safety critical and thus demand a high degree of assurance, however, developing these CPS is complicated by a number of factors including heterogeneity, unreliable network communication, mobility and a tight coupling with the physical environment [13]. In combination, these characteristics introduce a level of uncertainty of composition that is difficult to capture using traditional software methodologies. The special characteristics of CPS are reviewed below:

- **Heterogeneity** CPS demonstrate a high level of heterogeneity in terms of the devices that comprise the system. This may include:
  - Sensor nodes with a small amount of memory connected via low-bandwidth unreliable wireless networks.
  - Mobile devices such as smart-phones running over GSM-based technologies.
  - High-end workstations and servers connected via reliable wired networks.
This level of complexity demands rich support for the modeling of underlying technologies. This may include not only a variety of specification information, but also probabilistic models to predict factors such as the reliability of wireless connections or patterns of mobility.

- **Unreliable Networking** Many CPS applications operate over ad-hoc wireless networks and in power constrained environments. The strict power constraints of this environment preclude the provision of reliable networking, and thus communication in many CPS must be considered unreliable. Furthermore, the degree of unreliability exhibited by a CPS will vary according to environmental conditions. For example, a wireless network link may be reliable in good weather but fail during heavy rain due to absorption of microwave signals by rain water. These externalities cannot be modeled in a deterministic fashion and thus to be successful, any modeling approach for CPS must allow for the use of probabilistic models to predict network performance.

- **Mobility** CPS that incorporate mobile devices present an even higher degree of complexity. In mobile systems, devices may be forced to interact opportunistically as dictated by their communication range and unpredictable patterns of device movement. For example, in a vehicular network, interactions may be possible only when two vehicles come within range of each other. While in a subset of scenarios, the movement of mobile nodes may be predictable, in the majority of CPS, patterns of movement will be dictated by unpredictable factors such as movement of people, animals, or vehicles. As with unreliable networking, this necessitates consideration of probabilistic models to predict the movement of nodes.

- **Tight Environmental Coupling** Even in statically deployed CPS, a tight coupling with the environment means that system externalities require greater consideration than in traditional distributed systems. For example, a system deployed in an uncontrolled environment such as a flood plain [31] will be subject to a much greater variance of environmental conditions than a system deployed in a lab setting. For example, the wide temperature swings that occur from summer to winter, affect the speed at which crystal oscillators operate and must be considered if fine-grained time synchronization is required. Two approaches allow for the modeling of such environmental factors. Either, the performance of all system elements should be modeled in all possible conditions (though this would clearly lead to state explosion), or a feedback loop must be incorporated into system models that allows the behavior of the model to be adjusted based upon empirical observations of system behavior in different environmental conditions.

Due to these features of CPS, repeated analysis and incremental modeling are necessary, which copes with the features mentioned above and develops effective models of physical world, their properties, and interactions which should be linked with development methodologies. For some applications, an unsafe or incorrect functionality will lead to serious consequences such as loss of human life, and large scale environmental damage. Consequently a safety case must be crafted to argue why it is extremely unlikely that a single fault will cause a catastrophic failure. The removal of errors from the design and presenting a proof that the system meets the safety case is a great challenge to the scientific and engineering community.

### III. Research Challenges for Composition Mechanism in CPS

Cyber-Physical Systems (CPS) are inherently heterogeneous in their components, design requirements and infrastructure. Automated composition is a cost-effective technique of engineering such complex systems. In general, effective component-based system design depends on two properties: compositionality and composability. If these two features are not satisfied, the developed systems will not behave as expected and may be hard to maintain.

- **Compositionality** An application is compositional if the emergent behavior of the application may be derived from the behavior of its constituent parts. Compositional semantics thus provide a way to derive the meaning of any composition from its constituent parts. Therefore Compositionality modeling should be applied to ensuring the compositionality of components or services.

- **Composability** is a measure of the degree to which components can be assembled in various combinations to satisfy specific user requirements. We argue that run-time reconfigurable component models and SOA (Service-Oriented Architecture) offer optimal support for composability of application functionality, allowing for the composition to be changed at run-time.

Hence we argue that the composition mechanism in CPS involve two levels. At one level, composition modeling shall be used to ensure the compositionality of components during design phase. At the second level, application composition approaches shall be used to ensure the composability of systems during developing phase as well as at run-time. Main research challenges regarding the composition modeling of CPS include the following:

- **Composition Modeling for Functional Properties**: Static analysis and model checking have been widely used for systems to verify functional properties. Model checking is a generalized technique that is capable of searching very large state spaces for undesirable conditions and assuring that the system cannot reach such conditions. Static analysis and checking techniques for software are now widely used for bug-finding and more comprehensive analysis of industrial software. However in CPS, the behavior of systems may adapt to the change of external contexts, which in most cases are rich and non-deterministic. Therefore further research needs to be conducted to achieve comprehensive context analysis and checking capability for the wide range of interacting properties critical to system behavior.

- **Composition Modeling for Non-Functional Properties**: Further research needs to be conducted in CPS technology so that Quality of Service (QoS) guarantees can be
provided at the component level to guarantee emergent system-wide performance. The relative importance of security, privacy, robustness, inter-operability, extensibility, and mobility, as well as safety, must be specified and carefully verified.

- **Composition Modeling for Physical Properties:** In addition to functional properties, CPS are subject to a wide range of physical requirements, such as dynamics, power, physical size along with systems-level requirements, such as safety, security and fault tolerance. Therefore, it is critical that the research community develop modeling methods that take into account all of these requirements. CPS may have a wide range of physical requirements such as dynamics, power, physical size and memory etc, and thus the verification of these physical properties may vary. When components are integrated together, composition tools and approaches are needed to check and guarantee that physical properties are still satisfied.

- **Composition Modeling for Components Interfaces:** Research is needed towards checking the open systems interface. This should include tools (e.g., based on static analysis and model checking) that can check the interoperability of components and generate evidence for their composability and other properties.

- **Composition Modeling for Interface Coordination:** CPS are usually extensible systems, where components can join in ad hoc way. Current efforts to address interoperability are at the level of network connectivity. This will not be sufficient for future CPS. New systems will be required to orchestrate the interactions between devices and between devices and human users, and to assure that the device actions can be coordinated and carried out in real time.

In order to provide reliable and holistic composition models for CPS, the current fragmented methods must be replaced by much more inter-operable concepts for components, properties, and system representation. The models should have rich semantics including a shared semantic interface based on both physical and computational systems concepts. However, the class of models (computational, physical, properties) available today is impoverished relative to the systems we are seeking to build.

Furthermore, as different devices or users may join in the system in an ad hoc manner, the topology of the network may change at run-time, or application functionality needs to be updated due to changing environmental conditions or user requirements, the architectural modifications in CPS can occur at run-time. Therefore dynamic software architectures for CPS are necessary, which modify their architecture and enact the modifications during the system’s execution. This behavior is most commonly known as run-time evolution or reconfiguration [35]. Inspired by Bradbury [37], we list the reconfiguration challenges essential to CPS as follows.

- **Programmed reconfiguration** The change is triggered by the system and changes are defined prior to run-time. The requirement is decided by the nature of heterogeneity and environmental coupling of CPS as we described in Section II.

- **Ad-hoc reconfiguration** The reconfiguration is often initiated by the user as part of a software maintenance task, ad-hoc changes are defined at run-time and are not known at design-time. The requirement is decided by the nature of heterogeneity of CPS.

- **Constructible reconfiguration** The reconfiguration is often initiated by an external event from, for example, a user or an external monitoring tool. A description language is used to describe the initial system configuration. A modification language is used to describe architectural changes. The requirement is decided by the nature of heterogeneity and mobility of CPS.

- **Adaptive reconfiguration** It starts with a predefined set of configurations from the development stage of the software. The runtime dynamic changes are initiated by predefined events and then the system selects one of the predefined configurations and implements it. The requirement is decided by the nature of environmental coupling of CPS.

- **Constrained run-time reconfiguration** A change can occur only after pre-defined constraints are satisfied. For example, constraints can be placed on the architectural topology and the program state. The requirement is decided by the nature of unreliable networking of CPS.

Reconfiguration can be achieved through the parameterization, substitution, insertion, removal or reconnection of individual components. Runtime reconfigurable component models such as [5], [6] and [36] also support the reconfiguration of application compositions without requiring the suspension of application execution. However these models could not satisfy the above reconfiguration requirements of CPS. We need to seek more powerful methodologies though.

IV. COMPOSITION MODELING

Modeling and verification of CPS is complicated by their heterogeneous nature as well as their sheer complexity. A cyber physical system can be modeled by either a structural/architectural specification (how their components: sensors, actuators and processors work; and how they are interconnected together) or behavioral specification (showing the response of each component to an internal or external event). Existing modeling techniques for cyber physical systems rely upon semantics to represent the relationship between the cyber and physical features of a CPS, which is necessary for accurate modeling of any system. Generally speaking, mathematical formalisms (e.g. Multi-Mode Automata (MMA) [10], Timed process algebras such as [15], [16] and timed automaton [17], hybrid automata [26] and process algebras [27]) and description languages such as Labeled Hybrid Petri Nets (28)] are popular candidates for modeling CPS.

- **MMA** [10] is based on finite automata with different scheduling modes that are reflected in the different locations of automata. In this model, two-fold composition is used.
  - Hierarchical scheduling defines a tree where leaves represent elementary components and their associated schedules and non-leaf nodes represent a
composite component has its own scheduling policy under which the subcomponents are scheduled.

- Separate MMAs are used to define each sub-system and these are then composed together to form a whole system. Scheduling information is shared using a Multi-mode Resource Interface that abstracts over the internal events and shows only relevant information.

- Timed process algebras such as [15] and [16] and timed automaton [17] provide an abstraction of continuous behavior and describe scheduling explicitly using interleaved semantics to compose separate components. Hybrid automaton [18] and hybrid process algebras [19], [20], [21] combine continuous evolution and discrete changes to model systems behavior, and compose separate processes and automata using interleaving semantics.

- Hierarchical hybrid models, e.g. [22], [23], combine both parallel composition and agent architectures for modeling and composition of hybrid systems. More details on diverse compositional modeling techniques for hybrid systems can be found in [24], [25].

Process algebras have been widely and successfully used in a wide range of problems and in practical applications in both academia and industry for the specification and analysis of many different systems. Recently, through novel language constructs and well-defined formal semantics, several hybrid process algebras have been developed. They can be reasonably and effectively used to give formal specifications of various system designs including electronic system design [24]. In addition to the traditional simulation analysis of such designs, various sophisticated process algebraic analysis approaches or techniques (e.g. algebraic reasoning, formal verification and linearization algorithms) can be applied to such designs described in process algebra based formalisms. We list the following features of process algebras which lead process algebras to be promising methodologies for modeling and analysis of composition in cyber physical systems:

1) **Compositionality**: The ability to construct models in a compositional manner is a significant property when large complex systems are under consideration. Furthermore, it was already established that, for qualitative properties, analysis could also be compositional.

2) **Qualitative analysis**: For CPS, it is important to verify both correct functionality and timing properties. However, there is an issue of consistency when different formalisms and different techniques are being used for modeling for these two objectives. Process algebras allow an elaborated version of a functional modeling technique meant that the same system description (i.e. the same process algebraic specification) could be used to project models for both verification purposes.

3) **Mapping on automata**: Hybrid automaton theory and different extensions have been widely used for the analysis of CPS models. Over the decades, there are abundant hybrid automaton based verification tools (e.g. [29]) for hybrid systems. They have been successfully applied to address a variety of diverse industrial cases. In general, by means of straight forward translations (see [21] for details), hybrid process algebraic specifications can be translated to the corresponding hybrid automaton models. This enables verification of hybrid process algebraic specifications using hybrid automaton based verification tools.

While these approaches form a promising base for modeling and analyzing static CPS systems created using monolithic or OO approaches, these models do not incorporate functional requirements such as: scheduling, power consumption and security. In addition, these models do not provide sufficient mechanisms to manage the increasing dynamism of application compositions that is being driven by CBSE and SOA approaches. Extension of current approaches is therefore required if they are to support dynamic composition approaches such as CBSE or SOA.

V. APPLICATION COMPOSITION APPROACHES

Possible composition approaches for CPS include monolithic design, object orientation, application modules, component-based software engineering and service orientation. Section III.A to I.LE reviews each type of composition paradigm, focusing on the advantages and disadvantages of each approach in the context of CPS.

A. **Monolithic Design**

Monolithic application development approaches are inherently static, with application functionality set at compile time. Modifying system functionality in monolithic systems therefore requires wholesale replacement of the code-image and suspension of application functionality. Embedded C is a key example of a monolithic programming approach for embedded systems.

Monolithic approaches clearly offer limited scope for software reconfiguration as it is not possible to insert new functionality into the system after compile-time. Furthermore, without a clear modularization of systems functionality, any modification to the code-base carries with it the risk of unintended effects on unrelated areas of application functionality. The disadvantages of monolithic approaches are enumerated in more detail in [5]. Monolithic approaches are similarly poor in terms of their support for re-use. Even if a monolithic application contains reusable elements, there is no systematic way for a 3rd party developer to isolate and re-use these system elements.

A key advantage of monolithic design approaches is that they afford the possibility of static analysis prior to deployment of application functionality, this allows for exhaustive testing of system functionality [4]. In contrast, dynamic modularization approaches such as reconfigurable component models and service oriented approaches do not allow for exhaustive testing, as new application functionality may always be injected after deployment.

Monolithic design approaches thus offer a high degree of predictability, but little or no support for dynamism. As argued in Section II, dynamism is a key characteristic of CPS and
therefore monolithic approaches are particularly poorly suited for CPS in general. Nevertheless, monolithic approaches may be suitable for those CPS with very static requirements and that demand highly predictable application behavior.

B. Object Orientation

Object Orientation (OO) provides a form of static modularization of application functionality, wherein reusable objects can be composed together at compile-time to realize application functionality. However, OO approaches provide no support for the dynamic addition of application functionality after compile-time, and like monolithic approaches reconfiguration usually requires the wholesale replacement of the code image. Examples of OO approaches for CPS oriented approaches include embedded C++ [29] and Java ME [2].

Through modularization of systems functionality, OO approaches make it easier to re-use third party functionality across different application compositions. In addition OO modularization makes it easier to adapt and evolve existing software, reducing the tendency for changes in one area of application functionality to have unintended consequences in other areas of functionality. As application functionality is fixed at compile-time, OO approaches afford the same possibility for static analysis as monolithic design. However, they do not allow for runtime modification of a subset of system functionality.

When considering the dynamism and predictability trade-offs of OO design, it can be seen that OO affords the same opportunities for static analysis as monolithic design, while facilitating the modification and evolution of application functionality. However, OO approaches do not allow for the dynamic injection of new functionality at run-time. While this is a better fit with the characteristics of CPS than monolithic design, the level of dynamism remains limited.

C. Application Modules

The notion of application modules separates common, low-level system functionality such as process encapsulation, memory management and networking from higher-level application-centric functionality. Application modular composition approaches view system-level functionality as fixed and unlikely to change, while application modules are more likely to be dynamically deployed at run-time. Examples of development environments that support application modules include the Contiki Operating Systems [3] and the SQUAWK embedded Java Virtual Machine (JVM) [2].

By separating application functionality into dynamically deployed modules, application modular approaches allow for coarse-grained adaptation of system functionality over time. However, this is at the level of applications, which are themselves treated as indivisible units of functionality. In the case of Contiki [3] application modules are realized in C using a monolithic approach. In the case of SQUAWK [2], application modules are realized in Java using an OO approach. In all cases, updating application functionality requires the replacement of an entire module. Worse still, updating low-level system functionality requires the replacement of the entire Operating System image, causing significant disruption to the application at run-time. Furthermore, application modular approaches do not allow for the re-use of fine-grained functionality.

In contrast to Monolithic or OO approaches, the use of application modules imposes significant additional analysis overhead, as all combinations of application modules must be tested together to ensure reliable operation. In reality this is likely to be impossible, as new application modules may be introduced by third parties sometime in the future. In terms of adaptability, the potential to dynamically replace application functionality allows for coarse-grained system evolution without necessitating the suspension of application functionality and is thus a significant improvement over monolithic or OO design.

D. Component-Based Software Engineering

Component Based Software Engineering (CBSE) emphasizes the separation of concerns into self-contained black boxes of functionality which can easily be re-used [5]. In CBSE, all component dependencies should be modeled as explicit interfaces and components should have no implicit dependencies. This reification of dependencies allows for the possibility of automated compatibility checking between components. CBSE encompasses both static and dynamic component models as described below.

- Static Component Models: NesC [4] is perhaps the best known and most widely deployed component model for WSN and is used to implement the TinyOS operating system [30]. NesC provides an event-driven programming approach together with a static component model. The static nature of the component model means that unlike dynamic approaches, NesC components cannot be dynamically rewired to support reconfiguration. Thus modification of a NesC composition after deploy-time necessitates complete replacement of the code image on each mote and re-starting of the application. As with monolithic and OO approaches, the static nature of NesC allows for whole-program analysis and optimization [7]. However, unlike monolithic or OO approaches, developers have much richer support for re-using generic functionality between compositions, thus allowing for easier tailoring of functionality at compile-time, but not at run-time.

- Dynamic Component Models: OpenCOM [5] and RUNES [6] are examples of dynamic component models, which allow for the tailoring of application compositions not only at compile-time, but also at run-time, allowing for any aspect of system functionality to be dynamically updated without significant interruption the distributed application. OpenCOM is a general purpose, run-time reconfigurable component model that has been deployed in a number of CPS scenarios [31] [32]. OpenCOM features a small, platform independent run-time kernel and components may be developed using a variety of programming languages including C, Java and Python. The RUNES middleware [6] brings dynamic CBSE to
more embedded devices. As with OpenCOM, RUNES has been realized in both C and Java [3].

CBSE provides better mechanisms for the fine-grained re-use of application functionality than any of the approaches reviewed so far. In addition, Dynamic component models provide an excellent platform to manage changing application requirements and environmental conditions. However, the fluidity of application compositions and the ability to inject new code at run-time makes it essentially impossible to exhaustively test reconfigurable component based software prior to deployment. In addition, run-time reconfiguration requires some form of run-time kernel and therefore increases the overhead of these approaches.

E. Service Orientation

Service Oriented Architectures (SOA) build on the foundation of CBSE, extending the component model with semantic descriptions of services. Semantic descriptions allow for the discovery and re-use of 3rd party services at run-time. Until recently, SOA were considered to introduce too much overhead in CPS scenarios, which often contain embedded devices, however, recent work such as [33] and [34] have demonstrated that this may be feasible in some CPS environments.

Pohl et al. [33] apply loosely-coupled service bindings to the problem of industrial automation and control on a powerful sensor and actuator platform. Pohl argues that loosely coupled services are a promising model for binding components on sensor networks, as they promote interoperability, service discovery and allow for the re-wiring of bindings at run-time. REMORA [34] is a SOA for embedded systems that attempts to reduce the complexity of application development and service re-use through an event driven programming paradigm similar to that demonstrated by NesC [4].

Like CBSE, SOA allow for the creation of dynamic application compositions based upon re-usable units of functionality. However, through the provision of semantic descriptions, SOA goes a step further, allowing for the possibility of the automated discovery and re-use of third party functionality at run-time without the intervention of the developer. This represents the ultimate in flexibility and dynamism, however, it is extremely difficult to model, test and verify. In addition, SOA require both a run-time kernel to support reconfiguration and also the storage and transmission of semantic information within the network.

Table I provides a brief summary of the features of each composition mechanism for CPS.

The limitations of current application composition approaches become apparent when considering the process that a developer must follow to reuse a third party component. First the developer must discover the component through some form of search, which is usually based on plain-text descriptions of component functionality. Once a candidate component has been discovered, the developer must then read its interface declarations in order to understand how to bind the component into their application composition and then correctly parameterize the component. As no methods are provided to check that the component meets the requirements of the application composition, there is significant scope for error on the part of the developer. In addition, as components do not provide standardized information on their performance, it is difficult to assess their impact on the overall performance of the application composition. These problems effectively preclude the use of components that were unknown to the developer at application composition time and therefore render the automatic discovery and exploitation of new components impossible.

We propose to address these limitations by embedding semantic data in components that describes provided services, possible parameterizations and the performance of the component. This semantic data will allow for the automatic testing of compatibility between a component and an application composition. The presence of rich meta-data will also allow the developer to search the component repository based upon a detailed set of requirements. Semantic data will not only be available at development time, but also at runtime through introspection and will allow for the opportunistic discovery and re-use of third party components.

VI. TOUR GUIDE CASE STUDY

UPPAAL is a toolkit for simulation and verification of real-time systems. This tool can simulate and verify system modeled as networks of timed automata extended with integer variables, structured data types and channel synchronization, etc. The model-checker UPPAAL is based on the theory of timed automata and its modeling language offers additional features such as bounded integer variables, structure and functions. UPPAAL can clearly express the system model operation and check its property such as accessibility, safety and security [39].

However, due to the semantics limitation, UPPAAL could not express continuous variables, physical properties and other functional requirements such as: scheduling, power consumption and security. Therefore, we have reconstructed UPPAAL’S xml template files, added continuous variables declaration and related computation functions, developed the context computation functions which are essential for systems’ adaptation and the extended toolkit is called ComppaalGuider [38], as shown in Figure 3.
For the prototype of the tour guide system we mentioned in Section II, we have designed the four components as shown in Figure 4. Only one global clock is necessary and thus defined as $\text{guideclock}$ shown in Figure 5. Each component is described as follow:

- **LocationSensor** stays at the idle state and senses the location of a tourist when receiving a $\text{senselocation}$ message, as shown in Figure 6.

- **GuideProcessor** stays at idle state and sends $\text{senselocation}$ message to **LocationSensor** when receiving a $\text{adapt}$ message. After getting the location information and **GuiderContext**, **GuideProcessor** calls the function $\text{changed} := \text{locationChanged}(x\text{Loc}, y\text{Loc}, \text{GuiderContext})$ to see whether user’s location is changed or not. Next **GuideProcessor** sets up the X and Y dimensions of the location information and builds the tourist’s **LocationContext** and **ChangeContext**. Then **GuideProcessor** computes the **GuiderContext** and sends $\text{adapted}$ message to **GuiderAdapter**. The scenario is shown in Figure 7.

- **GuiderAdapter** needs to finish the context adaptation within 10 time units. **GuiderAdapter** first sends $\text{adapt}$ message to the **GuideProcessor**, asking the **GuideProcessor** to set up a new context. After receiving the $\text{adapted}$ message, **GuiderAdapter** calls the function $\text{adaptGuider}(\text{GuiderContext})$ and computes the tourist’s current location and sends the $\text{tailhedian}, \text{yangxindian}, \text{xiliugongornoguide}$ to the player. The scenario is shown in Figure 8.

- **GuiderAudio** plays the introduction when receiving the
right message.

The simulation process is shown in Figure 9.

We have also verified the availability, safety and liveness of the above model.

- **Availability:**
  - \( E \leftrightarrow \text{GuiderProcessor1.GetGuiderContext} \): the system can build the "GuiderContext" successfully.
  - \( E \leftrightarrow \text{GuiderAudio1.YangXinDian} \): the system can play the introduction successfully.
  - \( E \leftrightarrow \text{GuiderAdaptor1.AdaptSuccessful} \&\& \text{guideclock} < 10 \): the system must provide the run-time service within 10 time units.

- **Safety:**
  - \( A[\text{notdeadlock}] \): the system has no deadlock.

- **Liveness:**
  - \( A[\text{GuiderAdaptor1.TaiHeDian}] \\
  + \text{GuiderAdaptor1.YangXinDian} \\
  + \text{GuiderAdaptor1.XiLiuGong} \\
  + \text{GuiderAdaptor1.NoGuide} <= 1 \): the system can only have one judgement on the location of one tourist at any time.
  - \( A[\text{GuiderAdaptor1.YangXinDian} | \text{GuiderAdaptor1.TaiHeDian} | \text{GuiderAdaptor1.XiLiuGong}] \\
  \text{implyChangeContext[0].Val} == 1 \): the system can only play the introduction when the location of the tourist is changed.

The above verification results are shown in Figure 10.
in current composition mechanisms and highlight fruitful research directions. We propose that repeated analysis and incremental modeling are necessary, which cope with the features of CPS and develops effective models of physical world, their properties, and interactions which should be linked with development methodologies. To this end we use our toolkit ComposaalGuider for composition modeling during the design phase and LooCI middle-ware for composition approach during developing phase and at run-time and illustrate our approach using a tour guide case study. For our future work, we would like to investigate the tools and platforms for modeling and verifying composition in CPS through comparative case studies. Meanwhile, we would like to investigate the methodologies for describing interaction and collaboration among components in CPS and compositional security evaluation.

ACKNOWLEDGMENT

This work is supported by Research Development Funding received from Xi’an Jiaotong-Liverpool University (Project Number 2010/13).

REFERENCES


Ka Lok Man holds a Dr. Eng. degree in Electronic Engineering from Politecnico di Torino, Italy; and a PhD degree in Computer Science from Technische Universiteit Eindhoven, The Netherlands. Currently, he is a Senior Lecturer with the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, China and a research and engineering consultant for Solari, Hong Kong. His research interests include logic synthesis, simulation, formal verification and low power design methodologies for integrated circuits and systems, formalization of SystemC and SystemVerilog design, formal methods, process algebras, wireless sensor networks, specification and analysis of distributed, real-time, hybrid systems, embedded systems and physical cyber systems. On the above-mentioned topics, he has authored or co-authored more than 100 refereed publications.

Danny Hughes is a Lecturer at Xi’ an Jiaotong Liverpool University and a Free Associate of Katholieke Universiteit Leuven. He has published more than 65 academic articles and has received two best paper awards. Danny is chair of the IEEE Networked Embedded Systems for Enterprise Applications (NESEA) conference series and the ACM Middleware for Sensor Networks (MidSens) workshop series. Danny’s research interests are in the area of Wireless Sensor Networks, middleware for Networked Embedded Systems and Component Based Software Engineering. Danny holds a PhD from Lancaster University, UK.

Tomas Krilavicius is an associate professor at Vytautas Magnus university and Senior researcher at Baltic Institute of Advanced Technologies. He has published a number of papers and received one best paper award. Tomas is honorary member of Lithuanian Society of Young Researchers. He is a PC member of DATICS workshops and EURAS conference. Tomas research interests are in the area of formal modeling and analysis of Cyber-physical and Hybrid systems and application of Natural Language Processing for Information retrieval. Tomas holds PhD from University of Twente, The Netherlands.

Shujun Zou received her master degree from East China Normal University recently. Her research interests are in software engineering, context aware system and model checking. She has been a teaching assistant of Software Engineer and Software Testing. She has published 2 academic articles. Now Shujun Zou is a software engineer in China UnionPay.
Virtualizing Sensor for the Enablement of Semantic-aware Internet of Things Ecosystem

Sarfraz Alam, Mohammad M. R. Chowdhury, and Josef Noll

Abstract—The Internet of things (IoT) is a network of smart objects that bridges the gap between the physical and the digital world. The prevalence of IoT leads towards a new digital ecosystem for building novel applications and services. However, heterogeneity in devices, technologies and standards are causing to impede the widespread adoption of IoT. Advances in embedded service oriented architecture (SOA) for resource-constrained devices make it possible to expose the functional aspects of smart objects in the form of IoT web services. However, these IoT services are highly susceptible against a number of security attacks due to both the open nature and resource limitation of smart objects. This paper proposes a framework to address the service virtualization and security issues in order to ease off the IoT services development. Our exploration shows that the vigilant amalgamation of e-SOA, virtualization, semantic web technologies, sensor technologies paves the way to address the connectivity, security and monitoring issues of IoT.

Index Terms—Internet of Things, smart object, SOA, sensor, semantic technologies, security, virtualization

I. INTRODUCTION

WEB 3.0 is being transformed from connecting people and services to connecting objects (things). Today, a lot of devices and objects are emerged with sensors, enabling them to sense real-time information from the environment, and coupling this information with the web. This leads to a promising Internet of things (IoT) paradigm that allows connectivity of anything from anywhere at anytime. IoT creates a new digital ecosystem by amalgamating different technologies and standards, allowing different key players of industry to be part of it further. The realization of such paradigm requires implementing of Internet Protocol (IP), embedded web servers and web services stack (e.g., REST, SOAP) on smart objects and sensor nodes. Initially, it was conceived that IP is too heavy to be implemented on resource constrained devices. However, recent research studies [1][2] shows that it is possible to implement IP protocol even on real resource constrained devices. The advancement of resource constrained IP and its improved performance ensure scalability in IoT ecosystem. We already have witnessed IP scalability in the form the internet, connecting billions of computers today through IP. The IP is causing to bring convergence in IoT domain due to its scalability and ease of integration capabilities, and recently ZigBee alliance, announced to support the IP in their ZigBee enabled devices. Moreover, with such technology enablems, it is now possible to host a wide range of services on the resource constrained devices, enabling the inclusion of physical world as internet of things into the future internet. This coupling will not only open a new realm of novel and innovative services, but it will also brings new business opportunities for retail, logistics, food, health, energy, smart home, and transportation sectors. For instance, IBM utilized the IoT for Norwegian Sea oil platforms by implementing a service, which gathers real-time information from the bottom of Sea so that a better decision can be made in order to drill down to the Sea. Though IoT possesses benefits for society and business, but the full fledge realization is hindered by some critical technical challenges. First of all, IoT does not provide any registry mechanism for publishing service information publicly that is hosted on sensor. In [3], authors present a web-based repository, containing information about services from various sources and can be easily aggregated in the form of mashup. Secondly, different sensors and devices comprise with different data formats and models, thus causing IoT to exhibit deficiency in discovering and composing diversified services. Thirdly, IoT deficient in handling service invocation that sensor triggers with the occurrence of an event. The interaction between events and services are absent in current IoT clouds. Lastly, the integration of physical world into the fabric of the web imposes advance security requirements that need to be satisfied in order to ensure stringent control over IoT service interaction.

In this paper, we propose a semantic enhanced IoT virtualization framework to address the aforementioned challenges. The framework uses the Sensor-as-a-service (SenaaS) approach that does not only foster use of virtualization in IoT domain, but it also exposes the capabilities and the data of sensors in an IoT cloud in the form of services. In order to validate the proposed approach, we present a rail industry use case. The paper makes the following contributions:

- We propose a virtualized IoT framework, realizing the event-driven service oriented architecture (SOA) in IoT.
- We design ontology to contrive a semantic overlay of underlying IoT cloud.
- We present a security model to address authheincation and authorization in IoT.
- We present a policy-based service access mechanism and delineate polices in terms of semantic rules.

The rest of the paper is structured as follows. Section II discusses related work. Section III outlines sensor-as-a-service model. Section IV describes the proposed framework and

This work is in parts supported by the ARTEMIS pSHIELD Project and Norwegian Research Council.
S. Alam and M. M. R. Chowdhury are with Connected Life Research Group, University Graduate Center, Kjeller, Norway. E-mail: sarfraz@unik.no, mohammad@unik.no
J. Noll is with the University of Oslo (UiO) and the University Graduate Center (UnK), 2027 Kjeller. E-mail: josef.noll@unik.no
its details. Section V outline the service oriented security model to externalize the security operation in IoT ecosystem. Section VI, and VII addresses authentication and access control schemes in IoT respectively. Section VII presents proof-of-concept prototype implementation, including framework ontology, and the sensor platform used for the test deployment. Section IX outline the informal security analysis of the security proxy. We outline the objectives achieved by our design and some of the key open issues in section X.

II. RELATED WORK

Recent years have seen an explosion of IoT environment. It is anticipated as the next generation internet. Several research initiatives are going on in the area of IoT. However, IoT is tightly coupled with the sensor technologies because in most cases they provide the sensing, actuating and communicating capabilities to the IoT. Therefore, Sensor technologies have become pivotal to influence the physical objects in the creation and usage of future services. Example of such innovation services is Nike+iPod iPhone application [4], which records workout information and tracks burned calories with the help of accelerometer embedded into Nike jogger. The iPhone application acquires exercises and burned-calories information from accelerometer and presents on iPhone. Application also publishes this information on social network so that social network friends engage in challenges with each other.

Different EU projects, including [5] and [6] tried to develop a system that creates new services and applications for end users and enterprises by elevating the physical world to the digital world. However, these projects are successful to a certain extend to address issues such as heterogeneity, flexibility, sensor description and capability. Recently, Sensor Web Enablement (SWE) [7] provides a suite of standards, including sensor model language (SensorML) [7] that provides metadata model in XML format to describe sensor, its capabilities and measurement process. SWE model is inadequate in nature due to unintended meaning of XML and XML Schema. Semantic technologies have been used to provide intended meanings in many domain such as medical, telecommunication, search, to name a few. The use of semantic technology for capabilities description in mobile industry has already been explored in standardization [8] and research work [9][10]. Similarly, several system [12] and vocabularies [11] [13] have been developed for addressing the research challenges in sensor network domain. For instance, Horan et al. used the semantic technologies for describing capabilities of wireless transducer network [14].

Jie and Feng proposed a hierarchical structure sensor information system based on the semantic web technology [15]. They implemented common domain ontology to represent real-time data from sensors, but the ontology is represented in UML diagrams. In addition, the system is not flexible enough that could adapt according to different business scenarios and their changes. Haung and Javed proposed Semantic Web Architecture for Sensor Networks (SWASN), focusing on sensor data and inference over sensor data for wide range of WSNs [16].

Eid et al. used ontologies to formalize the sensors for searching distributed and heterogeneous sensor network data [17]. The study is mainly focusing on modeling sensor taxonomy hierarchy and sensor data ontology. However, the ontology lacks the description of sensor services.

The security of IoT is still in its infancy because most of the prior works are focusing on technology enablers for realizing IoT. Design principals and methods for securing IoT are yet to be explored. Recently, IETF start CoAP initiative [18] but they are just focusing on securing CoAP with IPSec and Datagram TLS (DTLS) algorithm. However, we are focusing on identity, access, and privacy issues of IoT. Reference [19] presented the overview of current security and privacy issues in the internet-of-things environment and proposed a personal policy-based privacy management system that automatically handles the privacy issues by incorporating the rules. Parikshit and colleagues proposed an identity management framework for internet-of-things that tries to tackle the issue of naming, addressing and resource discovery [20].

The security and privacy challenges of Internet of things that appear due to the involvement of diverse technologies are discussed in [21]. The author rates the sensitivity of those technologies based on the different security requirements such as confidentiality, integrity, authenticity, privacy, availability and regulation. The author mainly focuses on achieving confidentiality, integrity and availability whereas we also consider authentication, access control and authorization. The legal and ethical impact of the IoTs on the society have been discussed in [22][23] along with the technical aspect for securing IoTs. The appropriate legislations are required across the boundaries for the better utilization of IoTs, which provides awareness of privacy and ensures protection of user personal data in a legal and ethical way.

In [24], authors proposed web services based gateway approach to enforce the access control in mobile supported sensor networks using semantic web technologies. Reference [25] proposed several access control schemes using the one-way key chain and the Merkle hash tree for wireless sensor networks. Rui and colleagues presented a distributed privacy-preserving access control scheme for single-owner multi-user sensor networks [26]. They used prepaid token in order to access the sensor data. In [27], authors introduced the concepts of virtual identities (VIDs) to address the issues of security and privacy in the context of future internet-of-things.

Most of the aforementioned research works either focus only on vocabulary or sensor network framework. Whereas, we imply a holistic approach, considering the complete IoT ecosystem and provide a semantic overlay of underneath infrastructure that enable the framework to offer IoT capabilities and data in the form of web services.

III. THE SENSOR-AS-A-SERVICE (SENAA S) MODEL

The SenaaS model exposes functional aspects of sensor as services by hiding technical details of sensors from the user. The model assists in specifying, creating, managing, discovering and delivering sensor functionalities and capabilities as services. To date, sensor-world is facing a lot of challenges
such as sensor modeling, sensor orchestration, security and interoperability. However, we have seen that service-world has already experienced similar challenges on service level, and service-world has already developed a number of standardized solutions to address such challenges. Therefore, the key idea is to address the sensor-world challenges by lifting one step up and transferring them into service-world challenges so that one can exploit all the existing service-world standards to cope with sensor-world challenges.

IV. IOT VIRTUALIZATION FRAMEWORK

A. High Level View

The Fig. 2 depicts the high level view of the IoT virtualization framework. The framework targets the IoT cloud, consisting tens or hundreds of sensor, actuators, connected objects and devices instead of thousands of nodes. Furthermore, these nodes can be consisted of different classes of devices, including resource-constrained and resource-rich devices. The framework is capable of getting information from different sources and makes it available for novel services in from of virtual services. It also provides web service interface for functional aspects of smart objects of an IoT cloud. The main driver for the virtualization framework is re-usability of sensor information for a variety of novel service, both for owners and providers.

B. Functional Architecture

Fig. 3 depicts the functional architecture of the IoT virtualization framework.

The architecture is composed of three layers: (i) the Real-world access layer, (ii) the Semantic Overlay layer, and (iii) the Service Virtualization layer. It follows the event-driven SOA concept to address all the capabilities needed to respond to the dynamics of a real-time IoT infrastructure. SOA exercises a classic request/response communication pattern, and relationship between service and its consumer is synchronous due to the nature of request/response. This classic demand-based passive approach is not suitable for IoT clouds because all IoT clouds are event-driven that means events play a vital role in IoT cloud. Making sense of IoT events and performing a course of action in response to these events is the highly demanding capability of any IoT service framework. Here, we will provide the detail of each layer of IoT virtualization framework.

The real-world access layer provides an interface with underlying IoT cloud. It implies an adapter oriented approach to address the technical diversity regarding sensor types and communication mechanisms. One of the main goals of this layer is to get real-world information and carry it to the upper layer for further processing. It receives the sensor events and dispatches them to an event manager by using a callback message pattern, where messages are sent asynchronously to the receivers that later process the messages and take appropriate action. This layer can also transfers action messages from upper layer and then selects appropriate adapters to deliver it towards smart objects of an IoT cloud.

The semantic overlay provides the semantic model of underlying IoT cloud by maintaining an IoT ontology, the sensor ontology, an event ontology, and the service access policies. It facilitates CRUD (create, read, update and delete) operation on knowledge base. The layer is capable of importing any
sensor system description in SensorML and translates it to OWL description using the sensor ontology and the mediation rules, thus enabling automated reasoning capabilities over smart objects description. The use of SensorML ensures the interoperability with existing sensor infrastructure. This not only expose SensorML both physical and non-physical process as autonomous semantic-enhanced sensor services by incorporating domain knowledge but it can also enable automated service composition. The layer also supports both persistent and in-memory storage. The in-memory caching mechanism keeps the last observation of smart objects of an IoT cloud in order to boost the performance of the framework.

The goal of the service virtualization layer is to expose the functional aspects of smart objects of the underlying IoT cloud in the form of services. The layer aims at delivering requester the information they look for based on their access rights. The layer performs various tasks: It queries knowledge base for virtual service and translates them into semantic enhanced web service description. Further, it generates micro formats of available web services for publishing them on social network sites in order to increase the visibility of available sensor service. Through its service orchestrator, the layer can compose service based on the available services description. This layer is also responsible for notifying all the subscribers of a specific sensor event.

V. SECURITY IN INTERNET OF THINGS ECOSYSTEM

IoT is becoming omnipresent and IoT services are coming into reality. However, this realization is making the authorized access to IoT services and as one of the pivotal issue, which was considered to be trivial in the past due to very low number of attacks since these smart objects were operated in a close environment. Our earlier exploration shows that authentication, access control and privacy are the key security requirements that an IoT ecosystem needs to be satisfied [28]. Moreover, the study shows that the security models are not adequate to address the different security issue and they are outdated when employing in the service oriented context [29]. These contemporary security models can only provide point-to-point security, which does not fit in a IoT ecosystem that involves mediating services, and service proxies. We exploit the service oriented security proxy to satisfy seemingly divergent security requirements and externalizing IoT security operations. The security proxy consists of a policy enforcement point, an audit and a policy decision point. The proxy follows the principal of revers-proxy but we tailor it to the special characteristics of the IoT. This will allow security proxy as a special purpose intermediary service, abstracting the security related technical details (i.e., policy enforcement) from service consumers and allowing distributed provisioning of common security services. The security proxy acts as a service level gateway in front of IoT services. The service consumers access IoT service through the security proxy. In this case, the security proxy plays a role of edge-oriented policy enforcement point, which uses a policy decision point (PDP) to get access decision. The security proxy also maintains the log of all service call-outs that later helps in accessing security risks by evaluating logs.

To understand the service interaction and call-out, consider a Temperature Sensor Service that is associated with an IoT service called Smart Object Hosted Temperature Service. The service provides an interface to retrieve the temperature of a smart house. The security proxy receives request from the service consumer and retrieve the user credentials from the service request; afterwards it calls the service registry to get the associated real-world service description and then forwards this to the PDP for getting access decision that is required for proper access control. If access is granted the security proxy invokes Device Hosted Temperature Service and delivers the response to the service consumer. Fig. 5 illustrate the complete smart object hosted services life cycle.

VI. AUTHENTICATION IN INTERNET OF THINGS

The smart object hosted service invocation life cycle consists of two phases of authentication. In the first phase, the proxy identifies and authenticates the service requester before invoking the actual service. Since the proxy is a resource-rich device, any existing identity management solution can be integrated with the proxy. Whereas in the second phase, the smart object identifies and authenticates the service requests from the proxy. This outlook focuses on the second phase of authentication because the IoT environment associated resource constrained devices (smart objects), are not capable enough to communicate directly to the digital world due to the absence of full fledged IP stack. Therefore, we propose to establish a direct authentication scheme between proxy, acting as a client and smart objects acting as a server without any involvement of a third party or broker. This scheme enables smart objects to identify and authenticate the service request directly in a simple and uncomplicated way. Instead of employing static password based authentication method, providing low levels of security, we propose digital certificate and one time password (OTP) as the potential authentication methods that can be used interchangeably according to the smart objects capabilities and supported security features.

Fig. 4. Security proxy for externalizing the security operation of IoT
A. Certificate-based Authentication Scheme (CBAS)

This method leverages the X.509 digital certificate for ensuring authenticity of a message and the proxy. The certificate-based authentication uses public key encryption techniques that are backed up with public key infrastructure (PKI) especially for certificate management. However, we propose provisioning of trusted authorities certificates on smart objects, omitting the involvement of PKI from the smart object perspective. In this case, the proxy still needs to get the certificate either from the device provider or service provider in order to authenticate itself.

Here, the proxy and the smart object are two communicating parties. The proxy calculates a hash by taking the name of the service to be invoked, time-stamp, and device ID of the corresponding smart object as input parameters. It then encrypts the hash value with its private key and sends encrypted hash and the service name over to the smart object. Upon receiving, the smart object performs two tasks. First, it calculates the hash over service name, time-stamp and device ID. Second, it decrypts the received encrypted-hash with the proxy public key for obtaining the hash value send by the proxy. If the smart object’s calculated-hash and the received-hash from the proxy are the same, then both the message and the proxy is authenticated. The complete scheme is presented in Fig. 6.

B. OTP-based Authentication Scheme (OTPBAS)

This method leverages the HOTP algorithm for authenticating the proxy. The HOTP is based on an increasing counter value and HMAC-SHA-1, which involves static shared secret key between communicating parties. Therefore, we propose provisioning of supplying shared secret key to the proxy by some external means. The algorithm is as follows:

\[
\text{HOTP}(K, C) = \text{Truncate}(\text{HMAC-SHA-1}(K, C))
\]

We tailor the HOTP algorithm for the IoT ecosystem by shunning truncate function because this function is intended for human readable values whereas, this paper focuses on machine-to-machine communication. In order to authenticate, the proxy takes the name of the service to be invoked, time-stamp, device ID of the corresponding smart object and counter as input parameters to the HOTP
algorithm and then generates an OTP. After that, it sends the OTP and the service name over to the smart object. Upon receiving, the smart object also generates an OTP by taking same input parameters. It then compares the generated-OTP and the received-OTP, and if both are the same then the proxy is authenticated. Fig. 7 depicts the complete OTP based authentication scheme.

![Fig. 7. OTP based authentication scheme for authenticating the proxy](image)

VII. AUTHORIZATION IN INTERNET OF THINGS

Stringent security requirements of an IoT environment demand to enforce the proper access control for IoT services right out of the box. Traditionally, authorization logic is tightly coupled with the service logic, forming a rigid model where any change in authorization logic will require change in the service logic. Whereas, we propose externalizing of authorization that decouples authorization logic from the service logic. We see authorization in the form of policies that define service-level permission and privileges. The core of this approach is a PDP that is linked with a PEP and a policy information point (PIP). The user defines policies using a consistent format and consolidates them into the PIP, which works as policy repository. The PDP takes request from the PEP and then consults its polices for requested service. It will then make decision and return a response to the PEP. For instance, the security proxy may ask the PDP, "can Alice have invoke privilege for the Temperature Service". The PDP will ask for Alice credentials/attributes from the associated IdM and then evaluate them against existing polices for the service. If Alice happens to have invoke privilege for that service then the PDP will send an access decision of grant. If Alice does not have right privileges then the PDP will send an access decision of deny. This externalization of authorizations does not only eliminate the need for each service to make same set of access decision but also free the service developers from dealing the access decision making by decoupling the authorizations from the service logic. The complete scheme is presented in Fig. 8.

![Fig. 8. Access control protocol between the proxy and the PDP](image)

A. Case Study - Interoperable Rail Information System

The second use case of reference for this outlook is the continuously monitoring of trains and railway infrastructure. The purpose is twofold (i) detecting any unusual condition such as extremely high temperature, strange sounds and unexpected movement, and (ii) transferring and making available such information to different actors (i.e., train operator, rail infrastructure owner, consumer) involved in the rail system both automatically and in a request/response demand-based passive mode. The train, hereafter we will call smart train, is equipped with several heterogeneous computing devices such as sensors, actuators, GPS receiver, and gateway-embedded computer for detection of such conditions. These devices interact using heterogeneous protocols for sensing the information in their vicinity and send it to the gateway. As an intelligent device, the gateway figures out any irregularity, and it sends the details to the smart train operator. If the irregularity information is related with rail infrastructure, then the infrastructure owner and provider are also interested to know about such information. The gateway sends this information to all concerning actors, but they also need monitoring and periodically checking about the condition of the train and the rail infrastructure. The gateway embedded-computer is geared with the proposed IoT virtualization framework that exposes the smart train sensors as services for enabling such demand based and active remote monitoring.

B. Test Bed

The prototype system consists of an embedded Linux based gateway, 3G modem, embedded GPS receiver, and sensors connecting to the gateway. For sensors, we use the Sun Small Programmable Object Technology (Sun SPOT) platform, which is an OS-less, small, wireless, battery powered research platform. Each free-range Sun SPOT contains a processor, radio, sensor board and battery; the base-station contains only a processor and radio. The Sun SPOT uses a 32-bit ARM9 microprocessor with a clock speed of 180MHz, running the Squawk virtual machine. It has an integrated 2.4GHz 802.15.4 (TI CC2420) modular approved radio. In our implementation, we use built-in sensors (i.e., accelerometer, temperature) available with Sun SPOT sensor board. Furthermore, we also integrate FV-M8 GPS receiver to find the
location of the smart object. The gateway is geared up with IoT virtualization framework.

C. Framework Knowledge base

The Ontology, which is defined as formal and explicit representation of knowledge, is used for representing knowledge base. The ontology is a set of classes C, properties P and instances i. The key concepts of the domain are defined through classes. In ontology a property establishes a relationship between the two instances. A property belongs to a domain and has a range. Syntactically, a domain links a property to a class and range links a property to either a class or a data range [4]. From an instance point of view, a property relates instances from the domain with the instances from the range. The real actors of a practical use case scenario (e.g. individuals) are defined through instances and they belong to the classes. Among the different ontology languages [33], this work uses the Web Ontology Language (OWL) [34]. Fig. 9 illustrates breakdown of the knowledge base used here containing classes (nodes), subclasses, instances of classes or subclasses (is), properties (edge between nodes) and data values.

![Fig. 9. Breakdown of knowledge base, containing core concepts](image)

The ontology is developed around the core concept of IoT, which is composed of one or more instances of Connecte-dObjects concept, which in turns is composed of multiple instances of Sensor concept. The ontology provides shared vocabulary for describing concepts including Actors, Sensor, Sensing Capabilities and Virtual Services. It also defines a set of attributes (i.e., owl:DatatypeProperty) and relationship (i.e., owl:ObjectProperty), which holds between different concepts.

D. Access Polices

We propose using semantic technologies for policy definition. The policy is defined in terms of set of semantic rules that includes access rights, constraints and restrictions. The access control is mainly based on attribute/claim based access control. The attributes are also described using semantic technologies but the detail of this is beyond the scope of this outlook. The use of semantic technologies leverages the PDP with reasoning capabilities that helps in checking consistency of polices.

Table 1 described the authorized actors for each service. Following these assumptions, we formulated four different access policies that correspond four different types of service. The SWRL representation of the access policies are as follows:

E. Security Proxy

The security proxy runs as a web service on Apache Tomcat servlet container. It is developed using Apache Axis Web Service framework. The security proxy prototype consists of two core service: (i) authentication service that communicate with an external IDM system over Simple Object Access Protocol (SOAP), and (ii) authorization service that provides an interface for policy decision point. The service call-out is logged through a semantic enhance log service. For this, we developed a simple log ontology and service simply populate each service call-out as an instance of the log ontology. We formulated polices related with IoT services using web ontology language (OWL) and semantic web rule language (SWRL). We develop a simple user interface where end-user specify the high level policies, which later translate into semantic rules for accessing the IoT services. The prototype uses a RDF store for maintaining the user-defined polices. We use Pellet API [32] to exploit reasoning capabilities for evaluating the access policies and determining the access decisions by executing SWRL rules. The authorization service can be deployed either as an integral part of security proxy or an externally linked part of security proxy. The deployment decision is subjected to gateway device capabilities.

IX. SECURITY EVALUATION

The proxy and the smart objects communicate with each other such that the hosted-services are delivered to authorized end-user in a secure manner. The nature of the wireless channel opens a number of possibilities for an adversary to intercept the communication and modify the messages. Therefore, we focus on device-oriented goals (i.e., those concerning with proxy authentication) for the proposed protocol because the smart objects host number of services, which should not be accessible by any illegitimate entity. Although, adversaries can make use of different ways to make any scheme vulnerable against a number of security attacks, but a scheme can provide better defence by considering the number of potential security attacks at design time.

As smart objects are becoming prevalent, the exposure of these computationally low devices in the form of services will open a door for adversaries to target them easily and compromise their security. The adversary can launch a number of security attacks (e.g., man-in-the-middle, and replay) against smart objects to gain illegitimate access to their hosted-services and data. In this section, we analyze the proposed schemes by eyeing the aforementioned potential attacks and examining the overall security of the proposed schemes.

**Security Assumption:** an adversary receives a service request from the proxy and modifies the service request or the service name and relays the modified service request to the smart object.

**Attack:** man-in-the-middle with modification.
Countermeasure: The proposed CBAS attaches the signature with the service request. If the adversary modifies a single bit in the service request, then he needs to reflect that change in the attached signature. This requires the knowledge of all the input parameters used in the signature, and the proxy private key. The adversary does not know all the parameters and the private key. Therefore, the smart object easily detects any modification by the adversary during the signature verification and refuse to serve the service request.

In case of OTPAS, the adversary also needs to possess the shared key, counter so that he can modify the service request and relays to the smart object. When the smart object receives the modified request it first computes the OTP using the locally stored parameters that are only known to the smart object and the proxy, and then compare it with the received OTP. Both OTPs will be different because they are computed with different parameters. Hence, the smart object will determine the attack and will not honor the service request.

Security Assumption: an adversary somehow able to receive a service response from the smart object.

Countermeasure: the adversary cannot get any meaningful information from the message because the smart object sends the service response message in encrypted form along with the signature. The decryption process requires corresponding keys (private key, shared key), which are only known to the legitimate entities. It is extremely difficult to infer these keys from the encrypted message however, number of cipher text attacks can be applied. Moreover, the attached signature helps the proxy to detect any modification that can be occurred on the way.

Security Assumption: an adversary acquires the certificate from the smart object provider.

Countermeasure: in this case, the adversary holds the same root certificate that is installed on the smart object. Therefore, the smart object decrypts (step 1 in CBAS) the signature successfully. However, when the smart object verifies (step 2 in CBAS) the message hash, it will easily find the modification because both hash values are not same. In CBAS, the proxy maintains the each smart object device ID that is used while computing signature. If the adversary sends the service request to the smart object with a certificate from the same provider, the smart object will refuse to deliver the service response because the correct device ID is not included in the signature attached to the service request message. The adversary must be aware of device ID in order to launch such attack.

Security Assumption: an adversary captures the service request message and will retransmit it without any modification. Attack: replay without modification.

Countermeasure: as, a smart object receives a service request it first needs to detect the retransmission of the service request. If the smart object fails in detecting the retransmission, it then authenticates the adversary as a legitimate entity and sends the service response back to the adversary. Such retransmission can be detected by checking the received time stamp against the current time. If the service request time is older than the last successful service call, then the smart object will discard the service request. Being an intelligent device the smart object can also make an optimal guess about the service request initiation time by knowing the proxies distance and the travel time of the messages. The proxy also uses the same time stamp based procedure when it receives the service response message from the smart object.

In case of OTPBAS, the smart object utilizes the counter value for identifying the replay attack without modification. If the adversary attempts to replay the service request, the smart object will compute the OTP by using the locally stored parameters (i.e., counter value, shared key, etc.). The smart objects computed OTP is different from the received OTP since they hold different counter values. Hence, the smart object will invalidate the service request and initiate the counter synchronization notification towards the associated proxy.

Security Assumption: an adversary captures the service request message and will retransmit it after performing some modifications.

Attack: replay with modification.

Countermeasure: the smart object can discern any modification while validating the OTP received in the service request. In case the adversary replaces the time stamp, he also needs to replace the OTP if he wants to masquerade as the legitimate proxy. However, the generation of OTP is based on different parameters (i.e., shared key, counter, etc.) that are only known to the proxy and the smart object. Also, it is highly unlikely that the adversary can guess the correct shared key and counter values. When the smart object receives the modified service request it will compute the OTP using locally stored parameters and then compare the computed OTP against the received OTP. The comparison results in a negative response and therefore, the smart object will invalidate the

<table>
<thead>
<tr>
<th>Services</th>
<th>Polices represented by SWRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenaaS</td>
<td>Actør(?A) ∧ belongTo(?A, ?AT) ∧ SenaaS(?S) ∧ allowedTo(?S, ?AT) → canAccessTo(?A, ?S)</td>
</tr>
<tr>
<td>Monitoring</td>
<td>Actør(?A) ∧ belongTo(?A, ?AT) ∧ Monitoring(?M) ∧ allowedTo(?M, ?AT) → canAccessTo(?A, ?M)</td>
</tr>
<tr>
<td>Recommendation</td>
<td>Actør(?A) ∧ belongTo(?A, ?AT) ∧ Recommendation(?R) ∧ allowedTo(?R, ?AT) → canAccessTo(?A, ?R)</td>
</tr>
<tr>
<td>SOTA Advanced</td>
<td>Actør(?A) ∧ belongTo(?A, ?AT) ∧ SOTAAdvanced(?SA) ∧ allowedTo(?SA, ?AT) → canAccessTo(?A, ?SA)</td>
</tr>
</tbody>
</table>
service request.

Similarly, in case of the CBAS the adversary needs to modify the signature attached in the service request in order to incorporate any modification (i.e., timestamp, proxy certificate, service name) if he wants to replay the service request by masquerading as the legitimate proxy. Consider a sophisticated replay attack, where the adversary intercepts the service request and modifies the timestamp, the proxy certificate and signature. Once the smart object receives the modified service request it figures out the modification while verifying the signature. Because, the certificate attached to the service request does not contain the same root certificate that is installed on the smart object.

X. DISCUSSION AND FUTURE WORK

The proposed IoT virtualization framework demonstrate how sensor based services can be enabled in resource constrained smart object scenario. To realize this, the prototypical implementation used a rail industry use case. The paper follows e-SOA principle that adds complex event processing, allowing the framework to dynamically sense and respond to different events trigger by different smart objects in the IoT cloud. The combined effect of event-dispatcher and event-manager makes the system an active response based asynchronous system. Despite the slow uptick of applicability of semantic technologies in IoT domain, our solution tries to make existing IoT environment more interoperable using service virtualization and semantic description. With service virtualization the framework acts as entry point for the outer world and provides virtual endpoints for functional aspects of smart objects by hiding details of underlying sensor and actuator platform, hardware and software environment from the consumer of IoT services. Having semantic overlay of IoT cloud, it is very easy to query about the connected device and its services. However, the ontology used for the semantic overlay should be simple and lightweight in terms of complexity as we have seen the success of friend-of-a-friend (FOAF) [35] and semantically-interlinked online communities (SIOC) [36] in social networks.

The paper shows how security operations can be externalized through a proxy in resource constrained smart objects. The proof-of-concept implementation supports both the service consumer and the proxy authentication. In case of OTP-BAS, the out-of-band secret key distribution method eliminates the man in the middle attack for key distribution. Also, in case of highly resource constrained smart object the OTP based authentication can even be used without encryption. The scheme is vulnerable against a sophisticated re-play attack where the adversary jams the wireless channel and catches the number of OTPs when the proxy tries to invoke any smart object hosted service. These OTPs are valid for authentication until the counter value between the proxy and the concerned smart object is not synchronized again. However, this risk can be mitigated by synchronizing the counter value periodically. The OTP based authentication also poses the management of the counter and the secret key on the proxy side for each IoT environment smart object. In terms of performance, this approach only requires one calculation of hash function in order to authenticate the proxy.

The CBAS provides both the message and the proxy authentication. The smart does not support certificate revocation. However, the proxy acquires the certificate from the service/device provider, which could lead to confused service request problem. To exemplify, consider the residential apartments, where two users (e.g. user1, user2) are neighbors and both have the same device with the same service (i.e., temperature). In this case, both neighboring devices hold the same root certificate and if the proxy of the user1 requests the temperature service and if the smart object of the user2 receives the message then the smart object can consider it as a legitimate request. One solution to this problem is to introduce a personal certificate authority (CA) within IoT environment. Instead of such complicated approach, we rather employ a simple device ID based approach to mitigate the aforementioned confused service request problem. However, this fix comes at the cost of managing the ID of each smart object. In term of performance, this approach is more computational expensive because it requires two encryption and two hash calculation for authenticating the proxy.

We proposed semantically enhanced access control mechanism for deriving the access decisions. Over the years, researchers are working to bring the science of semantics to access control. The popular Role-based Access Control (RBAC) [37], Attribute-based Access Control (ABAC) [38], and Context-aware Access Control (CWAC) [39] models have already been semantically extended [40], [41], [42]. Adding semantics has been found to facilitate high level specification of access rights and constraints in the access control models. The use of semantic technologies provides flexibility to add complex constraints as well. Researchers lately supported the idea of using OWL and SWRL to represent policy [43]. Well-defined semantics, expressiveness of condition and extensibility are some of the crucial policy specification criteria [44]. According Coi et al., use of OWL and SWRL for formal specification of policies supports these criteria [44]. This semantic enhanced approach provides a more flexible and expressive way for defining and evaluating the IoT services policies. However, this comes at a cost of semantifi everything, meaning IoT services, policies, and attributes need to be described in a semantic way. This can be resolved by mirroring the service catalogue that is maintained by IoT virtualization framework. The performance is another major issue because reasoning over a knowledge and then executing rules are computationally expensive that could hampered the overall service request response time.

Our ongoing and future work includes a real time performance analysis of the proposed framework by deploying in a real environment.

ACKNOWLEDGMENT

The authors would like to thank Robert Engels from ESIS and Arne Dybdahl from Jernbaneverket for their valuable inputs and time.
REFERENCES


Sarfraz Alam is a PhD researcher at UNIK-University Graduate Center in Kjeller, Norway. He received his M.Sc. degree in the area of Information Security from The Royal Institute of Technology (KTH), Sweden. His research area covers Service Oriented Internet of Things, Semantic Services, SOA Security and Wireless Sensor Networks.

Mohammad M. R. Chowdhury received the Ph.D. from the Department of Informatics, University of Oslo in the area of security, privacy and trust. Before that he received the M.Sc. degree in Telecommunication Engineering from Helsinki University of Technology (HUT), Finland in 2002. His current research interests include access control, identity management, personalized service access, and security in Internet of Things. He is currently working as post doctoral fellow at University Graduate Center (UNIK), Kjeller, Norway. He was involved in research projects funded by the Norwegian Research Council and the European Union. Dr. Chowdhury has published about 35 scientific articles in journals, books and international academic conferences. He contributed in several international conferences as technical program committee member, session chair and reviewer. Dr. Chowdhury is a member of IEEE Communication Society.

Josef Noll is Professor at the UNIK-University Graduate Center and University of Oslo, Norway in the area of Mobile Systems. He is also Chief Technologist in Movation, an innovation company in Norway. He received his Ph.D. from University of Bochum, worked for European Space Agency at ESTEC from 1991 to 1997, and from 1997 to 2005 at Telenor R&D. His working areas include mobile authentication, wireless broadband access, personalized services, mobile-fixed integration, and the evolution to 5G system.
A Low Power and Small Die-Size Phase-Locked Loop Circuit Using Semi-Digital Storage

Markus Dietl and Puneet Sareen

Abstract—A conventional low-bandwidth Phase-Locked Loop (PLL) requires an external capacitor and a large on-chip ripple capacitor. A new PLL architecture is proposed in this paper, which replaces the large external capacitor in the loop filter by semi-digital storage cells. PVT compensation is achieved using the information stored digitally in the storage cells. Since the total value of the on-chip capacitor is reduced drastically, the proposed PLL architecture has a small chip size and a very low power consumption. The design is validated by a silicon implementation. The proposed architecture can also be extended to the design of high bandwidth PLLs.

Index Terms—Phase-Locked Loop, low power, real time clock

I. INTRODUCTION

In conventional low bandwidth PLLs [1], [2], [4], [5], the value of the capacitance in the loop filter gets so big that an external capacitance is mandatory. In addition to cost, off chip capacitors have the disadvantage of piezoelectric effects disturbing the PLL performance. Together with an off chip capacitor, a big on chip capacitor is needed (ripple cap) [2]. In order to reduce cost, there is a need to remove the off chip capacitor and also reduce the chip size. All digital PLL [8], [9], [10] has disadvantages due to quantization noise, power and complexity.

In this paper, a new PLL architecture is proposed, and the architecture is shown in Fig 1. The architecture replaces the big external capacitor in the loop filter by semi-digital storage cells, and each storage cell contains a charge pump (CP2). The reference current block (REFCUR) provides the reference current, which is mirrored to CP2 inside each storage cell. PVT compensation is achieved using the information stored digitally in the storage cells. Since the value of the on-chip capacitor is reduced drastically, the proposed PLL architecture has a small chip size and a very low power consumption.

II. PLL DEFINITION

A PLL [6], [7] can be perceived as a circuit synchronizing an output signal (generated by an oscillator) with a reference signal in frequency as well as in phase. In the synchronized state, often referred to as the locked state [7], the phase error between the output signal and the input or reference signal remains constant or is zero. However, if in the process due to some discrepancy a phase error builds up, a control mechanism gets triggered, which acts on the oscillator to counter-balance the phase error in such a manner that it is reduced to minimum until it is matched. The control system actually locks (matches) the phase of the output signal to the phase of the reference or the input signal. It is important here to note that, the PLL not only sets a fixed relationship between its output clock phase and the input or reference clock phase but also tracks the subsequent phase changes that are within its bandwidth. Due to its characteristics, the PLL is used for communication network systems and other circuits that require a clock recovery [8], frequency multipliers, and data synchronization.

III. CONVENTIONAL PLL

A basic block of PLL as shown in Figure 2 consists of five fundamental blocks, namely, phase detector, charge pump, loop filter, voltage controlled oscillator and the divider.

Phase Frequency detector (PFD) compares the phase of the periodic input signal against the phase of the feedback signal and accordingly generates a subsequent error signal which is proportional to the phase deviation between them.

The charge pump coupled to a phase detector block is a vital building block in the design topology. It helps provide an effective controlling mechanism for the charging and discharging of the low pass filter. It is used to manipulate the amount of charge on the loop filter’s capacitors depending on the lead (UP) and lag (DOWN) error signals generated from the PFD.

The loop filter included in this design architecture is a passive device comprising of two capacitors and a resistor. The output voltage generated from the loop filter informs the VCO to adjust (increase/decrease) its frequency in such a manner that the voltage output is maintained proportional to the charge of the capacitors.

Markus Dietl and Puneet Sareen are with Texas Instruments, Clock & Timing products, Freising, Germany. E-mail: m-dietl1@ti.com, p-sareen@ti.com
The voltage controlled oscillator is the most important device component of this feedback system that helps produce the essential frequency output (Fout) of the PLL. Depending on the control voltage of the low pass filter, the VCO generates the frequency that matches the reference signal.

When designing the PLL for very low bandwidth, the size of the loop filter capacitor gets so big that it is not possible to implement it on a chip. The external capacitor is needed in this case. The external capacitor increases the cost of the system in addition, it also increases the area on board which is not desirable especially in hand held devices. The capacitor’s piezoelectric effect also effects the final PLL performance.

IV. DESCRIPTION OF THE BLOCKS

A. Voltage/Digital Controlled Oscillator

The voltage controlled oscillator is shown in Fig. 3. A five stage single ended VCO is used here. The frequency of the oscillator depends on the current supplied by P11 and PN transistors. P11 PMOS provides a proportional gain. This gain is used to damp the PLL response. The voltage at PROP is biased to a reference voltage using two resistors R1 and R2. This control voltage is used for analog damping of the PLL. Here it is biased at VDD/2 (by choosing R1=R2, see Fig. 10).

VCO frequency is controlled with B[1: N] signals. When all B[1: N] are at high level, VCO will output the lowest frequency because these are connected to the gates of the PMOSes. While all are at low level, VCO will give highest frequency. So the VCO can have a very wide range depending on the number of programming bits chosen (B [1: N]).

B. Reference Current Block (REFCUR) and Storage Cells

The storage cell is the most important part of this PLL architecture. The VCO tuning range is divided into N small steps and N storage cells are used. The storage cell has two functions:

- Store the VCO tuning information digitally when being non active.
- Provide analog tuning when active, to enable a smooth transition from one digital step to the next.

Referring to Fig. 5, S1, S2,..., SN are the storage cells. The inputs LEFT and RIGHT are connected to the A outputs of the left and right neighboring cells. Inputs LEFT and RIGHT can have three possible combinations: LOW-LOW, HIGH-LOW and HIGH-HIGH.

The storage cell, which is shown in Fig.6, uses a small capacitance C1 to store the analog voltage corresponding to the running frequency of the VCO when being active. This capacitor of the storage cell is used only when it is active in the analog tuning mode. To achieve a smooth transition from one storage cell to the other, always two cells are active in analog mode, all other cells are inactive and the information will be stored digitally, using the PMOSes P2 and P3 to pull B HIGH, if both inputs LEFT and RIGHT are LOW.

If both inputs LEFT and RIGHT are HIGH, B is pulled to LOW using the NMOSs N12 in REFCUR (Fig. 7) mirrors the current to CP2 (N1) inside all storage cells via the net FAST. But it will have only an effect in the active cells if the inputs LEFT and RIGHT have different values (HIGH-LOW).

For these cells, B will be pulled down and VCO frequency will increase. If a DOWN pulse comes, the current will be mirrored to CP2 (P1) in all cells. B will be pulled up in all active cells, and VCO frequency will decrease. The time needed to charge/discharge can be controlled by choosing...
appropriate current mirror ratios. These currents should match to achieve low static phase offset. REFCUR determines these current ratios.

PMOS P6 in Fig. 6 initializes the capacitor at power up. Output A will depend on the voltage B at C1. If the voltage is above the threshold of the inverter (P5/N5) then the output is high, otherwise it is low.

When LEFT is HIGH and RIGHT is LOW, voltage B at C1 will neither be pulled up by P2 and P3 nor be pulled down by N2 and N3. The storage cell is active for analog tuning. At startup, all the storage cells (caps) are initialized to HIGH by N2 and N3. The storage cell is active for analog tuning. If the voltage is high, otherwise it is low.

As a result the PLL loop will try to increase the VCO frequency, the PFD will generate wide UP pulses and will discharge C1 inside S1. However, for S2 to SN, the RIGHT and LEFT inputs are LOW, because A [1: N] = 0. The corresponding storage capacitors (node B) are kept HIGH since P2 and P3 are on. As soon as node B in S1 goes below the threshold voltage (VDD/2) of IV1, node A in S1 will go HIGH and enable S2 for analog tuning. From this moment on, two consecutive storage cells are active and get affected by the UP/DOWN signals generated by the PFD. As soon as the output B falls below the threshold voltage of IV1 in S2, output A of S2 will go to VDD and S3 will be activated. Once this happens, RIGHT of S1 will get HIGH and N2 and N3 inside S1 are activated and voltage B at C1 is pulled to LOW. S1 is now inactive. The PLL loop starts filling 0’s starting from B [1]. The storage cells whose neighboring cell’s output A are at the same level are inactive and HIGH or LOW is stored digitally. Only the storage cells (two in this case) whose neighboring cells A outputs are at different levels, are enabled for analog tuning. At lock (ith update cycle) X-1 storage cells are inactive B=LOW and N-(X+1) are inactive B=HIGH.

\[
B [1 : (X - 1)] = 0 \quad \text{(3)}
\]

\[
B [(X + 2), : N] = \text{VDD} \quad \text{(4)}
\]

Storage cells with the Voltage B[X:X+1] are active, this means for the length of \(\Delta t_i\) of the ith UP or DOWN pulse, charge will be dumped at C1 (Fig. 6). This locking process can be visualized in Fig. 8.

Referring to Fig. 8, at the start when PLL is enabled (at time t(0), the net ‘B’ (Fig. 6) of all the storage cells are initialized to HIGH. That means ‘A’ of all storage cells is initialized to LOW. So PLL here will be operating at its lowest frequency and the first storage cell (S1 in Fig. 6) is enabled for analog tuning. t(1) is another point in time where voltage at ‘B’ inside storage cell ‘S1’ will discharge below the threshold of inverter IV1 (Fig. 6). At this point, the next storage cell (‘S2’) is activated for analog tuning. From this point on, two storage cells are always active for any analog tuning.

The time point t(x) shown in Fig. 8 shows another point in time where \(X^{th}\) and \(X+1^{th}\) storage cells are active for analog tuning and all other cells are held digitally either HIGH or LOW. Which will always be the case, that means PLL is either in locking process or already at lock.

Depending on the current and mirror ratio defined in REFCUR and C1 in the storage cell, it would take a pulse of the length of \(\Delta t_i\) to charge the voltage from VDD to GND. While B[X] and B[X+1] is active, the following equation is valid:

\[
B [X]_i = \text{vdd} \cdot \left( \sum_{\text{DOWN}} \frac{\Delta t_i}{dt} - \sum_{\text{UP}} \frac{\Delta t_i}{dt} \right) \quad \text{(5)}
\]

\[
f_i = K_{\text{prop}} \cdot V_{pi} + K_{\text{step}} \cdot \left( (X - 1) + \sum_{m=X}^{X+1} \frac{\text{vdd} - B [m]}{\text{vdd}} \right) \quad \text{(6)}
\]

where \(f_i\) is the average VCO frequency during ith update, \(K_{\text{prop}}\) is the proportional VCO gain (Fig. 3), \(V_{pi}\) is the average voltage at PROP during ith update pulse, and \(K_{\text{step}}\) is the frequency change per digital step.

Fig. 9 shows a SPICE simulation of the VCO frequency when CP2 is active continually (when UP is kept high). In this case B[1:N] (and A[1:N]) will start changing one by one.
The three vertical lines are A[30], A[40] and A[50]. Between A[30] and A[40], there are ten A’s(A31 to A40) from ten storage cells, changing from LOW to HIGH. These are not plotted in this graph. Similarly from ‘A40’ to ‘A50’ there are ten A’s(A41 to A50) from the next ten storage cells. The VCO frequency change with respect to the digital steps is very linear and smooth. There is no abrupt frequency change. This is possible only due to the fact that always two storage cells are active. The voltage handover from the storage cell getting deactivated to the storage cell getting activated is very smooth. The Voltage control Oscillator gain, from one storage cell to the next can also be controlled by the sizing of the corresponding PMOS transistor (Fig. 4). The purpose of that is also explained in Section V.

D. Phase Frequency Detector

A simplified circuit diagram of PFD is shown in Fig. 11. REFCLK and SYSCLK are applied to the clock terminal of the D-Flip-Flop (DFF). The D terminal of each flip-flop is connected to the logic HIGH. Rising edge on REFCLK/SYSCLK will set the corresponding DFF and reset the second flip flop after some delay (gate delay of OR gate in this case). Other PFD implementations can be referenced from [3] and [4].

V. PROCESS COMPENSATION

Process compensation is achieved using information in B[1:N] bits. Suppose VCO achieves target frequency at B[X]
in nominal conditions. That means B[1:X-1] are at low level and B[X+1:N] are at high level. Now as we create weak conditions, the VCO get slower and it needs more current to reach the same frequency. More B’s of the storage cells will discharge. So there will be more B’s at low level. Similarly, if there are strong conditions, VCO will be faster that at nominal conditions. So less B’s from storage cells are needed to reach the same target frequency. Also if the size of all the PMOSes driven by storage cells is same, then VCO frequency change per step will be more at strong conditions and will be less at weak conditions. So these individual PMOSes can be sized accordingly to keep the frequency change per step always same. All ‘N’ PMOSes will not have same size but scaled sizes with respect to the PMOS connected to B[X].

Similarly the change in current through CP1 and CP2 can also be controlled using the same information (either B[1:N] or A[1:N]) to keep it constant across corners. To do that, the value of the resistor (SR) inside REFCUR block and charge pump CP1 is scaled as shown in Fig. 12.

In order to increase the reference currents, more switches can be closed. This will short corresponding resistors, effective resistance will decrease, and finally the current will be increased. Similarly, in order to decrease the current, more switches can be opened. This will create more effective resistance and will reduce the current. Using this approach, we can compensate the PLL parameters over PVT.

VI. PROPORTIONAL DAMPING

In this semi-digital PLL, the oscillator is not controlled by a single voltage but by a proportional voltage and many storage voltages. The proportional voltage will be controlled similar to a traditional analog PLL. Fig. 13 shows traditional PLL where passive RC filter is used as a loop filter (for ease of understanding, ripple cap is not considered here).

When charge pump supplies current, a charge is deposited on the loop capacitor and is stored. On the other hand, resistor produces a voltage drop which is proportional to the current. The effect only exists when there is an UP/DOWN pulse. So this voltage changes the VCO frequency only for short duration and the effect is not stored. A ripple capacitor is commonly used to spread the effect of the resistor drop. The damping in the proposed PLL is shown in Fig. 14.

The loop capacitor, which was used to store the voltage for VCO, is replaced by a semi-digital storage (storage cells). For damping, a voltage divider is used to generate a reference voltage. As soon as the charge pump is switched off, the current is turned off and the voltage settles at VDD/2. The proportional voltage at the i\textsuperscript{th} update (referring to VDD/2) is given by

\[
V_{\text{prop}}(i) = I_{cp} \cdot \frac{R \cdot \Delta t_i}{T_{\text{update}}}
\]

where \(I_{cp}\) is charge pump current (CP1), \(R\) is the effective damping resistor, \(\Delta t_i\) is the phase difference at the i\textsuperscript{th} update at the PFD inputs, and \(T_{\text{update}}\) is the update period at the PFD input.

As shown in Fig. 1, the ripple capacitor is also used to spread the effect of the proportional voltage.

VII. MATHEMATICAL DESCRIPTION

Block diagram of the PLL is shown in Fig.15. Referring to Fig. 14, the semi-digital VCO (A/DCO) gets inputs from proportional path and from storage cells. So the PFD information (UP/DOWN) also goes to storage path consisting
of REFCUR and storage cells and proportional path consisting of charge pump ‘CP1’ and resistor divider.

\( T_{sys} \): Period length of \( i \)th period of the output clock
\( T_{ref} \): Period length of \( i \)th period of the reference clock
\( K_{prop} \): Proportional VCO gain
\( K_{step} \): VCO frequency change per step
\( R \): Damping resistor
\( Icp \): Charge pump current
\( t_{ref} \): The time of the rising edge of the \( i \)th period

\[
t_{ref} = \sum_{j=0}^{i} T_{ref} \tag{8}
\]

where \( t_i \) is the sum of all Periods before \( i \), i.e.,

\[
t_i = T_i + t_{i-1} \tag{9}
\]

PFD produces each update a pulse with the width \( \Delta t_i \). During this time, the charge pump current is flowing. So a charge will be deposited, with the size of

\[
Q(i) = Icp \cdot \Delta t_i \tag{10}
\]

After \( n \) updates the total deposited charge is

\[
Q(n) = \sum_{i=0}^{n} Icp \cdot \Delta t_i \tag{11}
\]

The capacitance translates that into voltage is

\[
V_{stored}(n) = \sum_{i=0}^{n} Icp \cdot \Delta t_i / C \tag{12}
\]

During PFD update (\( \Delta t_i \)), the charge pump current is flowing and produces a voltage drop over the damping resistor, with the size of

\[
V(i) = R \cdot Icp \tag{13}
\]

The rest of the update period no voltage drop is produced, so the average voltage drop (during one update cycle) is

\[
V_{prop}(i) = Icp \cdot R / T_{update} \tag{14}
\]

Fig 16 shows the VCO gains from proportional damping and storage gain. Here PN consists of \( N \) PMOS, controlled by \( N \) storage cells and P11 is the PMOS for proportional damping, each supplying current to the VCO. The working principle has been explained in Section IV-A. Storage cells ‘X’ and ‘X+1’ are the active storage cells where analog tuning is happening. All storage cells before ‘X’ and after ‘X+1’ are kept at logic ‘high’ or ‘low’ digitally. Cs is the capacitor inside storage cells. Icp_B is the current mirrored inside storage cells by the REFCUR block. Since two cells are active, the integration of the charge happens only in these two cells. The VCO frequency can be calculated as shown in Fig 17.

To filter the ripples produced by the damping resistor, a capacitor can be used (‘C’ in Fig. 1). This is shown in Fig 18. Now if the loop is broken in the feedback at the PFD input as shown in Fig. 19, then the PLL open loop transfer function is given as

\[
G(z) = (G_{prop}(z) + G_{store}(z)) \cdot \frac{N}{1 - \frac{1}{z}} \tag{15}
\]

\[
G_{prop}(z) = Icp \cdot \frac{N}{s \cdot C + \frac{1}{T_{ref}}} \cdot \frac{K_{prop}}{f^2} \tag{16}
\]

\[
G_{store}(z) = \frac{1}{1 + \frac{1}{z}} \cdot \frac{K_{step}}{f^2 \cdot dt} \tag{17}
\]
When the loop is closed, as shown in Fig. 18, then the transfer function is given as:

$$H(z) = \frac{G(z)}{1 + G(z)}$$  \hspace{1cm} (18)

The MathCAD plot of the closed loop transfer function is shown in Fig. 20.

VIII. SILICON IMPLEMENTATION

The proposed PLL architecture is realized with a 32.867 KHz real time input clock and a 38.4MHz output frequency in a 180nm technology. The chip is fully characterized, all the qualifications have been achieved, and the chip is already released to production.

The PLL bandwidth is set to 1KHz. The VCO is set to operate at a double of the output frequency (76.8MHz). The proportional gain is assigned to 15MHz/V and the step gain is 400KHz/step with CP1 current at 15\(\mu\)A. If CP2 current is constantly flowing, the time to complete one digital step (dt) is 63\(\mu\)s. Fig. 22 shows phase noise LAB measurement of the chip. Period Jitter and cycle to cycle jitter is less than 30ps and lab measurement snapshot is shown in Fig. 23. The layout of the chip is shown in Fig. 24. The PLL and its peripheral input/output stages consume 700\(\mu\)A current. The total Die area of the chip is 0.95mm².

IX. CONCLUSION

A new architecture without external components has been proposed for the very low bandwidth PLL application. The PLL architecture has a very low power consumption and a small chip size. The design has been validated by silicon implementation. The architecture is not limited to low Bandwidth
PLLs only, high bandwidth PLL can also be designed with very small die size, based on the same design principle.

REFERENCES


Markus Dietl was graduated from Technical University Munich, Germany in 1995. He started his career with Texas Instruments in Germany 1996. He worked for more than fifteen years in the field of PLL and clock drivers. He is working on clocking solutions for consumer and memory products. Main interests are modelling of PLL behaviour.

Puneet Sareen was graduated from National Institute of Technology, Jalandhar, India, in Electronics and Communication Engineering in 2000. He started his career with STMicroelectronics in India. He worked for more than four years in the field of programmable clock synthesizers. He joined Texas Instruments in 2005 and started working in the High Performance Analog Group. He is working on clocking solutions for consumer products.
A Novel System-Level Methodology for the Design and Implementation of Multiplexed Master-Slave System-on-Chip Components using Object-Oriented Patterns

Sushil Menon and Suryaprasad Jayadevappa

Abstract—In a customer driven environment, systems pose a great challenge to designers in terms of complexity and time-to-market. In the past, designers have looked at Commercial-Off-The-Shelf (COTS) based development techniques that rely on integrating components produced by various manufacturers. However, the vast range of components from various manufacturers leads to obscurity of interfaces between them, making system integration a dreadful task. Hence, large warehouses of intellectual-property (IP) modules comprising of various software simulation models of system components at different levels of abstraction that can be readily integrated to form a Virtual Prototype are highly beneficial towards the exploration of various architectures.

In order to design and implement IP modules efficiently, we identify the Design Patterns that they follow. We propose a Design Pattern based methodology for modeling and implementing multiplexed Master-Slave devices at the system-level. As an effort to showcase the methodology, we choose to model a Level-2 Cache (L2Cache) using SystemC. A technical specification document of the Motorola MPC2605 L2Cache is procured and modeled using hierarchical state machines that we implement using SystemC. The developed model is an integrated look-aside, no write-allocate L2Cache supporting 4-way set-associative cache mapping and LRU replacement algorithm, compatible with the Simple Bus. The L2Cache module is then further parameterized to accommodate changes in L2Cache size, cache mapping strategies and replacement algorithms.

In this paper, we present the proposed methodology through a description of the process of modeling and verifying an L2Cache IP module at system-level, using the Master-Bus-Slave/Master-Bus-Slave Design Pattern. The test-bench involves a master (to generate transactions), a Simple Bus, the L2Cache Module and Slave Module (Simple memory). Various relevant test cases were generated to test the functionality of the developed module. The test-case outputs were routed to a debug file which is inspected to determine whether the output is as expected. We conclude by enunciating our learning from our experience with this methodology.

Index Terms—Electronic System Level, Level-2 cache, Intellectual Property, design patterns, Transaction Level Modeling, SystemC

I. INTRODUCTION

Ever inflating user requirements have been posing a challenge to the designer in terms of increasing system functionality. Today’s typical cell-phones have additional features like cameras, MP3 players, Internet browsers and even televisions. To support these features they comprise of about more than 80 peripheral devices [2]. Further, cell-phones are estimated to contain over a million lines of code [1]. Increasing system complexity in terms of hardware and embedded software add to the difficulties of system design. System Architects are further strained with diminishing time-to-market. The estimated time to convert a modern day cell-phone from concept to product is typically about 18 months [2].

COTS has provided the interim relief to handle the current issue of complexity and time-to-market. However, the increasing sources of components lead to an increase in the probability of an architectural mismatch, thereby making systems difficult to integrate, and also increases the cost of system integration. Hence, System Architects need to move up the value chain, using newer tools and methodologies that are superior to the current method of madness of designing systems. Electronic System Level (ESL) design is an effective alternative for System Architects, enabling them to combat the complexities and requirements of systems to come, also supporting architectural exploration or “What if?” scenarios very early in the design cycle.

ESL design deals with designing systems at higher levels of abstraction. Higher levels of abstraction increase system comprehension by capturing the functionality of a component and eliminating unnecessary details that are required for system synthesis, thereby providing high-speed simulation models that are suitable for the verification of software applications. The Electronic Design Automation (EDA) industry uses several tools, including Rosetta, SystemVerilog and SystemC for modeling systems using ESL design. SystemVerilog (IEEE standard 1800) is used in the industry for hardware design and verification at the system-level. It is an extension of the Verilog Hardware Description Language (HDL) which supports the design and constrained random verification of parameterizable hardware modules wherein random test-vectors are input as a stimulus to the design under test. The results of the simulation are automatically verified using language assertion features
called SystemVerilog Assertions. SystemVerilog also supports the usage of functional-coverage features that measure the completeness of the testing performed, and constraint features that direct test-vectors towards boundary test-cases, thereby increasing the test-coverage.

Although SystemVerilog supports system-level hardware design and verification, its HDL roots do not permit the co-design and development of hardware and software applications which are typically developed using Object-Oriented software languages such as C++. Hence, there is a need for a modeling platform which supports ESL design and enables the simultaneous design and development of hardware and software.

SystemC (IEEE standard 1666) is a commonly used platform that supports ESL design and enables hardware-software co-design. It is an open-source event-driven simulation kernel written in C++ that supports modeling at various levels of abstraction including Transaction Level Modeling (TLM). TLM allows designers to model systems at a higher level of abstraction, by separating component functionality from inter-component communication. Thus, TLM is highly suitable for implementing Virtual Prototypes at higher levels of abstraction.

Virtual Prototypes are software simulation models of the hardware components of a system that provide the complete functionality of the system, thereby enabling the usage of software simulation models during early phases of system architecture and design, hence aiding architectural exploration – “What if?” scenarios. Virtual Prototyping provides functionally accurate models of the hardware, allowing software development at early stages of system development, and also provides System Architects with a platform for analyzing performance statistics such as power estimation at the system-level [5]. Figure 1 illustrates a typical Virtual Prototype – Virtio’s VPXS Virtual Platform. It comprises of an instruction-accurate instruction-set simulator (ISS) of the Intel XScale core, functional models of the respective peripheral system-components and a cycle-accurate model of the on-chip bus which is implemented in SystemC [8].

However, there exists a class of integral system components, such as L2Caches, DMA Controllers etc., that exhibit multiplexed Master-Slave functionality. Such system components adopt the Master-Bus-Slave/Master-Bus-Slave pattern, where Slave/Master represents the multiplexed Master-Slave device. Figure 4 illustrates the Master-Bus-Slave/Master-Bus-Slave pattern and Figure 5 shows the sequence of messages exchanged in a typical Master-Bus-Slave pattern based system. In normal-mode, the multiplexed Master-Slave operates in a manner identical to that of the Master-Bus-Slave pattern. However, in multiplexed-mode, the multiplexed Master-Slave operates in a manner identical to that of a transaction router. First, the Master initiates a transaction...
which is forwarded over the Bus to the multiplexed Master-Slave. On receiving the transaction from the Master, the multiplexed Master-Slave routes the transaction over the Bus to the Slave. The Slave then forwards its response over the Bus to the multiplexed Master-Slave. On receiving the response from the Slave, the multiplexed Master-Slave routes the response over the Bus to the Master, thereby completing the transaction.

![Diagram of Master-Bus-Slave/Master-Bus-Slave pattern](image1)

**Fig. 4.** The Master-Bus-Slave/Master-Bus-Slave design pattern

In this paper, we propose a methodology based on the usage of the Master-Bus-Slave/Master-Bus-Slave pattern, whereby multiplexed Master-Slave IP modules can be modeled efficiently at the system-level. We present the methodology through a description of the process of modeling and verifying an L2Cache IP module, using the specified Design Pattern. This paper is organized as follows: First, we provide a brief description of previously related work, highlighting our motivation and our contribution. Next, we formally present the proposed methodology. We then demonstrate the proposed methodology through the process of modeling, implementing and verifying an L2Cache at Transaction Level using SystemC. Finally, we conclude by enunciating the advantages of the proposed methodology and our learning from our experience with this methodology.

**II. RELATED WORK**

Modern electronic systems are characterized by increasing integration complexity, thereby necessitating novel design methodologies that allow architectural exploration at reduced integration costs. Platform Based Design (PBD) is one such design methodology wherein systems are composed of fully-parameterizable computational and communication elements, defined at an appropriate level of abstraction, thereby minimizing integration costs and aiding architectural exploration. Vincentelli in [1] defines a Platform as “a library of components that can be assembled to generate a design at that level of abstraction”. Thus, the success of PBD is highly dependent on the easy access to these large libraries of IP modules.

Several authors have considered the use of Design Patterns for efficient and extensible hardware design. Doucet and Gupta in [9] introduce the bus-protocol pattern for specifying on-chip bus structures and associated protocol behaviors, and the DLX processor pipeline pattern for modeling a DLX pipelined processor. Astrom et al. in [10] use the composite, object adaptor, abstract factory and decorator patterns to model a DSP library. Damasevicius et al. in [11] introduce the wrapper pattern that permits the behavioral adaptation of an IP module to the requirements of its operational environment, which they use to generate handshake wrappers using the single-rail-4-phase handshake protocol. MacKay in [12] discusses the application of the planar, grid-on-chip and symmetric-multiprocessing patterns in the design of multicore embedded systems.

Despite the plentiful work done in exploring the application of Design Patterns for hardware design, there still exists an unexplored yet promising domain – the proposition of a Design Pattern based methodology for the efficient design and implementation of IP modules. Although Doucet and Gupta [9] present a methodology based on the usage of Design Patterns for the translation of a model of computation into hardware, neither do they discuss its applicability to the design and implementation of IP modules, nor do they demonstrate its usage. Our work aims at addressing the need to design IP modules efficiently by exploring the usage of Design Patterns in the design cycle. Our contributions are as follows:

- We propose a methodology based on the usage of a pre-existing Design Pattern – the Master-Bus-Slave/Master-Bus-Slave pattern – whereby multiplexed Master-Slave IP modules can be efficiently modeled and implemented at the system-level.
- We demonstrate the proposed methodology by modeling and implementing an L2Cache IP module using SystemC as our choice of ESL design language.

**III. THE PROPOSED METHODOLOGY**

We now present a methodology based on the usage of Design Patterns whereby multiplexed Master-Slave IP modules can be modeled and implemented efficiently at the system-level. The methodology is based on a pre-existing Design Pattern – the Master-Bus-Slave/Master-Bus-Slave pattern – and uses this pattern to filter out a component’s structure and
functionality from its communication interfaces. It consists of the following seven steps:

1. Capture the requirements tabulated in the module specification using a Use-case diagram.
2. Derive the structure of the module and its logic using a module block-diagram and finite state machines (FSMs).
3. Using the Master-Bus-Slave/Master-Bus-Slave pattern, filter out the module’s structure and functionality from its communication interfaces.
4. Implement the module block-diagram and FSMs using SystemC.
5. Implement the module’s communication interfaces and the system test-bench using SystemC.
6. Partition the requirements represented in the Use-case diagram into equivalence classes and draft suitable test-cases using the partition testing technique.
7. Using the system test-bench, verify the implemented module using the previously drafted test-cases in order to ensure that it is developed precisely.

In the following sections, we demonstrate the proposed methodology through the design, implementation and verification of an L2Cache IP module at Transaction Level, using SystemC as our choice of ESL design language.

IV. DESIGNING THE L2CACHE AT TRANSACTION LEVEL

An L2Cache is a small, high-speed memory located between the Processor and main system memory (i.e. RAM), that is used to improve system performance. Frequently accessed data is placed in the L2Cache, thereby allowing the Processor to access the data at a higher rate than RAM. However, when the requested data cannot be located in the L2Cache, the Processor is stalled for additional penalty clock-cycles equivalent to the number of clock-cycles to retrieve the data from RAM and store it into the L2Cache. Thus, System Architects are constantly researching the effects of L2Cache parameters (such as L2Cache size, replacement algorithms and cache mapping strategies) on system performance. Since L2Caches are located between the Processor and RAM, they are multiplexed Master-Slave devices (Slave to the Processor, Master to the RAM), making them complicated to model, as some transactions include both Master and Slave functionality. An L2Cache IP module is developed and tested as per the specifications of the Motorola MPC2605. This device is an integrated look-aside, no write-allocate L2Cache supporting 4-way set-associative cache mapping and Least Recently Used (LRU) replacement algorithm, compatible with the 60x Bus. The MPC2605 is an L2Cache, designed by Freescale Semiconductor, for use with processors that implement the PowerPC architecture.

A. Elaboration of Requirements

The L2Cache IP module has to support parameterization in order to aid architectural exploration. The requirements to be supported by the L2Cache IP module are:

1. L2Cache sizes: ranging from a minimum of 1KB to a maximum of 4GB.
3. Replacement algorithms: Least-Recently-Used (LRU), First-In-First-Out (FIFO) and Random.
4. Support only for Random replacement algorithm in-case Fully-associative strategy is chosen, since support for LRU or FIFO replacement would involve a huge overhead to implement the counters that track the LRU or FIFO blocks.
5. Ignore the choice of replacement algorithms in-case Direct-Mapping strategy is selected.
6. Specification of a replacement algorithm is mandatory in-case Set-associative strategy is selected.

These requirements are represented with the help of a Use-case diagram shown in Figure 6.

B. Deriving the L2Cache Block Diagram and Modeling the State Machines

This phase involves understanding the technical specification and translating verbose, textual information from the specification, into a suitable module block diagram and hierarchical state machines that are desirable for implementation purposes. Figure 7 shows the module block diagram of the L2Cache that we derived from the specification.

As shown in Figure 7, the L2Cache module consists of the following components:
• **Data-Ram array**: this is used to store the data that is frequently accessed by the Processor.
• **Tag-Ram array**: this is used to store the tags of the addresses, whose data is stored in the Data-Ram array.
• **L2Cache Controller and Bus Interface unit**: this unit implements the state machines of L2Cache and the interface for inter-component communication.

As mentioned earlier, the L2Cache module is designed using hierarchical state machines. Hence, some state machines contain states that are sub-state machines, making the framework highly modular. Taking advantage of the modular framework, we modified minor portions of the state machines and extended the MPC2605 architecture to include the parameterization features, namely different cache mapping strategies, replacement algorithms and L2Cache sizes. The top-level state machine of the L2Cache is shown in Figure 8.

![Fig. 8. Top-Level state machine of the L2Cache module (states with a * denote sub-state machines)](image)

**C. Capturing the Module Communication Interface**

In this phase, we capture the L2Cache module communication interface using information acquired from the message sequencing chart of the Master-Bus-Slave/Master-Bus-Slave pattern. The communication interface, as shown in Figure 7, consists of a Control-Bus which channels the control-word that specifies the class of a transaction (read/write), an Address-Bus which channels the memory-address of the corresponding transaction, and a Data-Bus which channels the data associated with the corresponding transaction. Since the L2Cache is a multiplexed Master-Slave device (refer Figure 5 for the sequence of messages exchanged), the channels of the communication interface are modeled as bi-directional channels (represented as double-ended arrows in Figure 7), thereby allowing the L2Cache to generate and respond to transactions. An interesting observation to note is that although the module block-diagram (Figure 7) suggests an explicit Control-Bus channel, due to the knowledge of the class of the transaction currently occurring in the system, we choose to model the Control-Bus channel implicitly as shown later.

**D. Implementing the Derived State Machines using SystemC**

This phase involves translating the hierarchical state machines into SystemC code. First, the general structure of the L2Cache IP module is captured and translated into a SystemC module. Figure 9 shows the SystemC code that captures the general structure of the L2Cache IP module.

As shown in Figure 9, the Data-Ram and Tag-Ram arrays are captured as arrays of SystemC Bit-Vectors. Each state machine of L2Cache is captured as a C++ function/method. Hierarchical state machines are implemented by invoking the respective sub-state machine function from within the current state machine function. The Bus-interface functions are responsible for linking the L2Cache module to the communication infrastructure provided by SystemC (which will be covered in the next sub-section). Finally, the L2Cache module is initialized through the invocation of the respective initialization method from within the module constructor.

![Fig. 9. SystemC code capturing L2Cache Module structure](image)

![Fig. 10. SystemC code capturing an L2Cache state machine structure](image)

Next, each state machine modeled from the previous phase is captured using a C++ method/function as shown in Figure 10. Each method begins with a list of states in the respective state machine. The starting state is set by equating a state-marker variable to the initial state. The entire state machine logic is embedded in an infinite loop, switch-case C++ construct as shown in Figure 10, with each case-label denoting the respective state of the state machine. Within each case-label, the logic for the respective state is implemented using SystemC code. Shifting between states is accomplished by equating the state-marker variable to the name of a new
state defined in the list at the beginning of the method. Sub-state machines are executed by invoking a C++ method/function of an internal state machine function listed in the module structure code.

E. Implementing the Module Communication Interface and the System Test-bench

The proposed test-bench consists of a Processor (a master that generates the necessary transactions), the L2Cache and RAM, communicating over the Processor and Memory buses, as shown in Figure 11. The Simple-Bus is an open-source communication infrastructure, shipped with SystemC that provides interface methods to attach communicating components. These interface methods accept the memory-address of a transaction and the data associated with the transaction as arguments, and hence we use them to model the Address-Bus and the Data-Bus channels of the L2Cache. The Control-Bus channel is implicitly captured by the provision of discrete read/write interface methods, thereby eliminating the requirement of an explicit channel. The class of the current transaction is determined by identifying the interface method that is invoked during the transaction. Integrating the L2Cache into the test-bench is accomplished by attaching the L2Cache to the Processor and Memory Simple-Buses, which involves the following procedure:

1. First, the Simple-Bus slave-read and slave-write interfaces of the L2Cache are implemented to invoke the L2Cache read state machine and write state machine respectively, as shown in Figure 12.

2. Next, the Bus-master port of the L2Cache is connected to the Memory Simple-Bus and the Bus-slave port of the Processor Simple-Bus is connected to the L2Cache, as shown in Figures 13 and 14. The “Top” module instantiates the system test-bench objects and links them together to form the virtual system.

Since all modules are modeled at Transaction Level, every transaction between the Processor and L2Cache takes exactly one clock-cycle of the SystemC engine, with the Processor executing on the positive edge of the clock and the L2Cache and RAM (if accessed) executing on the negative edge of the clock. Since the L2Cache module is a multiplexed Master-Slave device, it must perform all execution and return the results to the Processor (the master) on the negative clock edge. Hence, it is ensured that there are no wait-states during the L2Cache execution.

F. Extending the L2Cache for Parameterization

There exist a multitude of parameterization techniques that designers can choose from, some of which include the C++ Traits mechanism. However, since the focus is on methodology, we opt for a more primitive, conditional compilation technique; a brief description follows.

Each state machine is analyzed for changes that occur due to the selection of a different parameter. The changes made to the state machines are then translated into modifications in SystemC code by encapsulating each modification within suitable conditional compilation directives. Thus, by defining the respective pre-processor symbols, the module can be parameterized to select any of the different features supported i.e. changes in L2Cache size, changes in the cache mapping strategy and changes in the replacement algorithm. Figure 9
shows an example of how conditional compilation directives are used to compile the Data-Ram and Tag-Ram arrays differently for the N-way Set-associative mapping strategy, thereby supporting parameterization.

V. EQUIVALENCE CLASSING OF MODULE REQUIREMENTS AND DERIVING THE TEST-CASES

This phase involves the equivalence classing of the module requirements represented in the Use-case diagram (Figure 6) by partitioning the module configuration parameters into disjoint subsets, each of which exhibit the same behavior, thereby allowing the selection of a single representative test-case from each class, hence alleviating module verification by reducing the effective number of test-cases. Figure 15 illustrates the segregation of the equivalence classes of the L2Cache module requirements into three domains, namely Mapping Strategies, L2Cache Sizes and Replacement Algorithms.

Figure 15. Equivalence classes of the L2Cache module configuration parameters

Exhaustive testing involves the verification of all combinations of module configuration parameters, leading to the possibility of twelve test-cases, some of which are inappropriate due to the selection of an invalid combination of parameters, namely:

- Verification of the Fully-associative strategy with LRU or FIFO replacement algorithms or with no replacement algorithm is deemed illegal due to the violation of requirement 4, eliminating three test-cases.
- Verification of the Direct-mapping strategy with LRU, FIFO or Random replacement algorithms is deemed illegal due to the violation of requirement 5, eliminating three test-cases.
- Verification of Set-associative strategy with no replacement algorithm is deemed illegal due to the violation of requirement 6, eliminating one test-case.

Having identified the inappropriate test-cases, we ignore these seven invalid combinations of configuration parameters and utilize the partition testing technique, thereby reducing our verification domain to the remaining five valid combinations shown in Table 1. The following section showcases the process of verification of each of these five valid test-cases.

<table>
<thead>
<tr>
<th>Mapping Scheme</th>
<th>L2Cache Size</th>
<th>Replacement Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>Any</td>
<td>None</td>
</tr>
<tr>
<td>Fully-Associative</td>
<td>Any</td>
<td>Random</td>
</tr>
<tr>
<td>Set-Associative</td>
<td>Any</td>
<td>LRU</td>
</tr>
<tr>
<td>Set-Associative</td>
<td>Any</td>
<td>FIFO</td>
</tr>
<tr>
<td>Set-Associative</td>
<td>Any</td>
<td>Random</td>
</tr>
</tbody>
</table>

Table 1. Valid module Configuration Parameter combinations

VI. VERIFYING THE L2CACHE IP MODULE

The process of verification involves implementing the previously derived test-cases as a series of transactions initiated by the Processor, triggering the appropriate functionality of the L2Cache. In the following sub-sections, we present each of the five derived test-cases that showcase the process of module verification.

A. Test-case #1: Verification of the Direct-Mapping Scheme

We set the L2Cache size to 1KB and select the Direct-mapping scheme. The logic of the test-case is to replace the same block in L2Cache by only changing the tag. Reading from the replaced address should result in a Read-miss since the L2Cache is Direct-mapped. In order to implement the test-case, we design the Processor to generate the following transactions in the given order:

1. 1st Transaction: Write Data 0x12345678 to Address 0x2A8. This will result in a Cache-miss, thereby writing the data to RAM.
2. 2nd Transaction: Read from Address 0x2A8. This will result in a Cache-miss, thereby fetching the data from RAM and storing into Block 0xAA in L2Cache.
3. 3rd Transaction: Write Data 0x6AC5 to Address 0x6A8. This will result in a Cache-miss, thereby writing data to RAM.
4. 4th Transaction: Read from Address 0x6A8. This will result in a Cache-miss, thereby replacing Block 0xAA with data 0x6AC5.
5. 5th Transaction: Read from Address 0x6A8. This will result in a Cache-miss, thereby replacing Block 0xAA with data 0x6AC5.
6. 6th Transaction: Read from Address 0x2A8. This will
result in a Cache-miss since Block 0xAA now holds data with respect to tag 0x1 whereas Address 0x2A8 results in tag 0x0.

Figure 16 shows the waveform generated from the previous set of transactions. Notice that from time 15ns onwards, the Replaced Block # has been set to 0xAA, indicating that the L2Cache has replaced the same block. Also notice that at time 55ns, L2Cache signals that there has been a hit (the 5th transaction) and at time 65ns, L2Cache signals that there has been a miss (the 6th transaction). This successfully verifies the behavior of the Direct-mapped L2Cache.

B. Test-case #2: Verification of the Set-Associative-Mapping Scheme with LRU Replacement Algorithm

We set the L2Cache size to 1MB and select the 2-way set-associative mapping scheme, along with the LRU replacement algorithm. The logic of the test-case is to bring in data into the 2-blocks of a given set and access the 1st block, hence reflecting that the LRU Block is the 2nd block. In order to implement the test-case, we design the Processor to generate the following transactions in the given order:

1. 1st Transaction: Write Data 0x12345678 to Address 0x2AAA8. This will result in a Cache-miss, thereby writing the data to RAM.
2. 2nd Transaction: Read from Address 0x2AAA8. This will result in a Cache-miss, thereby replacing Block #0 of Set 0xAAAA in L2Cache.
3. 3rd Transaction: Write Data 0x23468971 to Address 0x12AAAA8. This will result in a Cache-miss, thereby writing the data to RAM.
4. 4th Transaction: Read from Address 0x12AAAA8. This will result in a Cache-miss, thereby replacing Block #1 of Set 0xAAAA in L2Cache.
5. 5th Transaction: Read from Address 0x2AAA8. This will result in a Cache-hit, hence resetting the LRU counter of Block #0 in Set 0xAAAA in L2Cache.
6. 6th Transaction: Read from Address 0x22AAA8. This will result in a Cache-miss, thereby replacing LRU Block #1 in Set 0xAAAA in L2Cache.

Figure 17 shows the waveform generated from the previous set of transactions. Notice that from time 45ns onwards, L2Cache signals that the LRU Block is Block #1, hence replacing Block #1 during the 6th transaction. Also, notice that at time 55ns, L2Cache signals that a read-hit has occurred (5th transaction). This successfully verifies the behavior of the 2-way set-associative, LRU replacement L2Cache.

C. Test-case #3: Verification of the Set-Associative-Mapping Scheme with FIFO Replacement Algorithm

We set the L2Cache size to 1MB and select the 2-way set-associative mapping scheme, along with the FIFO replacement algorithm. The logic of the test-case is to bring in data into the 2-blocks of a given set and access the 1st block, showcasing that, unlike the LRU replacement algorithm, since the FIFO Block is the 1st block, it will always be the choice of replacement, irrespective of the access to a block. In order to implement the test-case, we design the Processor to generate the following transactions in the given order:

1. 1st Transaction: Write Data 0x12345678 to Address 0x2AAA8. This will result in a Cache-miss, thereby writing the data to RAM.
2. 2nd Transaction: Read from Address 0x2AAA8. This will result in a Cache-miss, thereby replacing Block #0 of Set 0xAAAA in L2Cache.
3. **3rd Transaction**: Write Data 0x23468971 to Address 0x12AAA8. This will result in a Cache-miss, thereby writing the data to RAM.

4. **4th Transaction**: Read from Address 0x12AAA8. This will result in a Cache-miss, thereby replacing Block #1 of Set 0xAAAA in L2Cache.

5. **5th Transaction**: Read from Address 0x2AAA8. This will result in a Cache-hit, which would reset the LRU counter of Block #0 in Set 0xAAAA of an L2Cache implementing the LRU replacement algorithm, but will have no effect on an L2Cache implementing the FIFO replacement algorithm.

6. **6th Transaction**: Read from Address 0x22AAA8. This will result in a Cache-miss, which would replace LRU Block #1 in Set 0xAAAA of an L2Cache implementing the LRU replacement algorithm, but will replace Block #0 in Set 0xAAAA of an L2Cache implementing the FIFO replacement algorithm.

Figure 18 shows the waveform generated from the previous set of transactions. Notice that from time 65ns onwards, L2Cache signals that the FIFO Block is Block #0, hence replacing Block #0 during the 6th transaction. In contrast, at time 55ns, L2Cache signals that a read-hit has occurred (5th transaction). This successfully verifies the behavior of the 2-way set-associative L2Cache.

### E. Test-case #5: Verification of Fully-Associative Mapping

We set the L2Cache size to 4KB and configure it appropriately so that it is organized as a heap of 1024 cache-blocks. We also select the fully-associative mapping scheme, thereby allowing accessed data to be located in any of the 1024 cache-blocks. The logic of the test-case is to show that multiple memory-addresses that map onto the same block in a 2-way set-associative L2Cache do not introduce conflict misses, indicating that the L2Cache is fully-associative. In order to implement the test-case, we first initialize all 1024 blocks of the L2Cache to store valid data for 1024 unique memory-addresses that would map onto the same set in a 2-way set-associative L2Cache. We then design the Processor to generate 1024 read transactions to the previous memory-addresses in order to query the Hit-Status of the L2Cache.

Figure 19 shows the waveform generated from the previous set of transactions. Notice that each of the 1024 read transactions that query the Hit-Status of the L2Cache results in a Cache-hit, indicating the L2Cache is fully-associative. In contrast, for a 2-way set-associative L2Cache, since each of the memory-addresses maps onto the same set, only the last 2 read transactions that query the Hit-Status would result in a Cache-hit. This successfully verifies the behavior of the fully-associative L2Cache.

### D. Test-case #4: Verification of Fully-Associative Mapping

We set the L2Cache size to 4KB and configure it appropriately so that it is organized as a heap of 1024 cache-blocks. We also select the fully-associative mapping scheme, thereby allowing accessed data to be located in any of the 1024 cache-blocks. The logic of the test-case is to show that multiple memory-addresses that map onto the same block
transactions results in the replacement of a random block within set 0x2A. This successfully verifies the behavior of the
16-way set-associative, random replacement L2Cache.

VII. CONCLUSION

The application of Design Patterns in the design of electronic systems promotes architectural re-usability via the
isolation of computational logic from communication infrastructure, thereby allowing designers and systems
architects to re-use communication affiliated architectural infrastructure, enabling them to focus exclusively on
modeling the required computational elements that comprise the system. In addition, the usage of Object-Oriented design
and testing methodologies and engineering tools such as UML, use-cases and state-machines empowers designers and
System Architects with an efficient and effective engineering alternative, resulting in the production of high-quality
electronic systems.

In this paper, we present a methodology whereby electronic systems can be efficiently designed and implemented using
Design Patterns, which we then demonstrate through the design and implementation of a multiplexed Master-Slave IP
module at the system-level. The increasing utilization of Virtual Prototyping in the manufacture of electronic systems
coerces the design of parameterizable IP modules and the creation of IP warehouses that provide designers and System
Architects with a variety of components designed at various levels of abstraction that can be readily integrated. We believe
that the generation of IP modules would be accelerated through the adoption of a systematic and efficient methodology
such as the one proposed in this paper. The efficiency of the proposed methodology is attributed to the usage of readily available, pre-defined and pre-verified Design Patterns that permit architectural re-usability. The systematic nature of the proposed methodology is attributed to the seamless flow offered between successive phases in the methodology. The usage of such a methodology provides designers and system architects with effective tools that would allow them to combat the complexities and time-to-market requirements of systems of the future.

REFERENCES

Robust Optimization and Reflection Gain Enhancement of Serial Link System for Signal Integrity and Power Integrity

Jai Narayan Tripathi, Raj Kumar Nagpal, and Rakesh Malik

Abstract—System level signal integrity and power integrity problems for high speed serial links have been explored in this paper. An example of the USB 2.0 IP has been used in this paper, but the analysis is generic for all serial links. This paper considers signal and power integrity as effects simultaneously. A model is developed to optimize the performance of high speed serial link in terms of jitter and amplitude performance. Sensitivity analysis is carried out with a set of dependent parameters affecting the performance. Taguchi array optimization has been applied during the optimization process. Finally, reflection gain concept is also applied to further improve the performance for the eye diagram. A strong correlation between measured and simulated results is shown. A generic methodology for SI and PI for high speed serial links is presented with complete analysis of package, board, termination, squidd card, decoupling network etc.

Index Terms—Signal integrity, power integrity, serial links, bit error rate (BER), high speed data transmission

I. INTRODUCTION

Signal Integrity (SI) and Power Integrity (PI) are the most critical issues as semiconductor industry is moving towards higher operational speeds. Signal integrity and power integrity should be looked at system level rather than looking at active and passive networks separately. System level analysis becomes a necessity when the individual subsystems work according to specifications, and even after that complete system does not work accordingly. System Level Simulations of an IP are necessary to ensure its reliability and robustness. Since the speed of the integrated devices is increasing in an exponential manner, Signal Integrity (SI) effects and Power Integrity (PI) effects the system performance and system bit error rate. In high speed digital devices, Signal Integrity (SI) and Power Integrity (PI) are the most important factors for the designers to keep in the mind while designing a system, as they affect the reliability of transmission at high data rates. This research work intends to analyze SI and PI for On-the-Board System emphasizing on reliability and robustness analysis of PHY IP for high speed data operations. As the industry is growing, the speed of the systems is increasing but the voltage supplied is being reduced. There are more and more complex routing structure being used. This complication at higher speeds results in reflection, crosstalk, EMI and other high frequency effects. In semiconductor industry package analysis, board analysis, mixed signal simulation etc. are performed separately. All the subsystems are working properly does not mean that the complete system will work properly, according to the specifications. Thus there is a hard need for an integrated simulation to ensure the proper functionality of the system. This paper presents a generic flow for complete on-the-board system level simulation to simulate and analyze the reliability and robustness of any PHY (with example of USB 2.0 PHY), in context of high speed data transmission. On-the-Board system level simulation means integrated simulation of die, package and board (for the sake of simplicity, this will be represented by ‘system level simulation’ throughout the paper). A flow is set for complete on-the-board system simulation which can be used for the pre layout verification within robustness limits of the IP. Universal Serial Bus(USB) 2.0 PHY IP is used for this work but the methodology is generic for all the serial link systems. Three advantages of SI and PI Analysis are:

1) This analysis is useful to perceive the behavior of whole system at simulation level accurately.
2) This can be used to ensure the robustness and reliability of a system.
3) It will help the designers to modify the system before its fabrication thus helping to reduce product cost.

II. HIGH SPEED ISSUES: A SYSTEM PERSPECTIVE

With the increasing speed of the systems, the time for a signal to travel from one end of an interconnect to the other end which is called propagation delay, is becoming considerable portion of the bit unit interval. High speed can be defined as the range of speed when the length of the interconnect is in comparable to the wavelength of the signal propagating through it. The content of energy in frequency domain also informs about the degree of deformation in signal in time domain for a given bandwidth of interconnect. The maximum frequency content of a signal can be found by its rise/fall time. A thumb rule to calculate the maximum frequency content of a signal [14] is

\[ f_{max} \approx \frac{0.35}{\tau_r} \]  

(1)

Above the \( f_{max} \), the energy is negligible. Thus taking in to account the interconnect length, clock speed and signal slew rate, we can predict whether the shape of the signal will be considerably distorted or not. Such high speed systems...
and their interconnects require a lot of resources and intensive computations for simulation so Model Order Reduction (MOR) is used [16]. MOR takes into account only dominant poles of the passive circuits.

A. Signal Integrity

Signal Integrity (SI) is an emerging area for the research. As speeds of the systems is increasing, designers have to think more about it. “There are two kinds of designers, those with Signal Integrity (SI) problems and those that will have them” [64]. Signal Integrity is concerned with the quality of the signal at high speeds. SI can be defined in a better way as, “Signal Integrity ensures that a signal is moved from point A to point B with sufficient quality or integrity to allow effective communication” [14]. For High Speed transmissions to maintain SI, bandwidth of interconnects should match the bandwidth of the system [14]. Signal Integrity analysis becomes important to ensure reliable high speed data transmission. SI is more dependant on channel at higher frequencies. With increasing speed, there are various challenges to maintain SI through channel optimizations [4]. A channel causes different types of losses in a signal. Channel loss reveals what fraction of voltage transmitted by transmitter will be received by receiver [7]. Signals come from real world i.e. from outside the chip. There are various packaging levels through which the real world signals have to pass. These interconnects are mainly responsible for the SI problems. Signal distorts with time due to the physical length it has to propagate through. There are different mechanisms by which signal distorts but the following are the main reasons [5]:

1) Insertion Loss: Insertion Loss is the loss due to the propagation of the signal. The signal gets significantly degraded after a certain distance. This loss is due to the IR drop, skin effect, dielectric losses etc. At higher frequencies, ohmic losses are more due to uneven current distribution. Conductance losses are proportional to dielectric loss factor which also depend on the rise and fall times of the signal [16]. Due to the degradation in voltage, signal can not retain its shape and Signal Integrity is affected. Insertion loss of a channel can be calculated by the ratio of received signal to the transmitted signal, which is generally represented in dB.

2) Crosstalk: Crosstalk is the coupling between the two lines in a close proximity. Coupling between two lines depends upon the spacing between them, the metal of the lines and the widths of lines etc. [63]. Crosstalk occurs all at the levels, at chip level, at package level as well as at PCB level. Crosstalk affects the system performance in two ways. First, it changes the effective characteristic impedance of the coupled or the victim line which results in timing delay of the signal. Second, it adds noise in the adjacent signals [34]. There are two types of crosstalk, ‘Near End Crosstalk’ and ‘Far End Crosstalk’. Near End Crosstalk is generated at the nearest point of the victim line, from where the signals starts traveling and the Far End Crosstalk is generated at the farthest point where signal travels in the coupled or victim line. Near end crosstalk depends on the signal (rise/fall time) if the delay of the system is greater than it, while far end crosstalk always depends on the edge rate (number of edges) of the signal. The very typical way to characterize crosstalk is to measure root mean square value of the coupled signal on a quiet net [7].

3) Reflection: Reflection occurs when there is a discontinuity in transmission path. These discontinuities cause impedance mismatch, most of the time. The very common discontinuities in high speed transmission path are connectors, wire bonds, vias, solder balls etc. Characteristic impedance of a transmission line is the ratio of the electric field to the magnetic field at a point. If a transmission line with characteristic impedance $Z_0$ is connected to the another one of characteristic impedance $Z_0'$, the reflection coefficient between them is:

$$ \Gamma = \frac{|Z_0| - |Z_0'|}{|Z_0| + |Z_0'|} $$

If $Z_0' > Z_0$, we get an overshoot in the signal and if $Z_0' < Z_0$, we get an undershoot in the signal. In Section 7, this concept of overshoot is used to improve the Eye Diagram.

The above three effects Insertion Loss, Crosstalk (Coupling) and Reflection are generally represented by S-Parameters (Refer to Figure 15).

4) Inter Symbol Interference (ISI): Inter-Symbol interference (ISI) primarily appears due to lossy characteristics of transmission line which results in frequency dependent behavior [65]. The maximum degradation due to ISI appear in walking 1 or walking 0 type of patterns. Multiple bits get merge in one level eliminating the fast variation due to single bit change. ISI is the reflection of rise/fall time degradation of the signal as it travel down the interconnect media.

B. Power Integrity

Power integrity term covers proper design of power delivery network. Quality parameters defining Power delivery network are power supply ripple, power supply spectral performance for the targeted switching load. A good definition of Power Integrity (PI) is, “Power Integrity deals with the power delivery paths from a voltage source to active devices (ICs) through boards and packages” [15]. There are three main performance indices of a power delivery system [15]:

1) Sufficiency
2) Efficiency
3) Stability

Sufficiency means ability to deliver sufficient power to the targeted switching load. Efficiency means to switch on/off the power as per load requirements. Stability means capability to maintain stable power supply to designated Devices (chips) at required times when noises from other parts of the design interfere [15]. While designing PDS, we should take care of power distribution in chip, power distribution in package and power distribution in boards. These designs become critical at higher switching loads as maintaining lower target impedance at higher frequencies require sufficient decoupling network. The power delivery network modeling involves true 2D/3D power plane impedance profile model extractions. Two main noises, introduced due to PDS are SSN and ground bounce noise.
1) **Simultaneous Switching Noise (SSN):** Signal nets should be modeled as transmission lines [62]. Assuming, a power plane supplies power to 50 numbers of drivers. If 30 of them switch at a time, the current drawn from the power plane will be very large so voltage fluctuation is created in power supply. A large amount of current drawn abruptly from the power plane results in a \( \frac{di}{dt} \) noise which is equal to \( I_{peak} \frac{di}{dt} \). The worst case becomes when all the drivers are ON [62]. At signal level, the quiet signal is affected by many signals so it does not become quiet. To reduce SSN, there are some suggestions by [34], some of them are:

1. Design different power supplies for core circuit and I/O circuit.
2. Maximize the on-chip capacitance. This will work as a charge carrier or a battery and will supply the current.
3. Place the board capacitors as close as it is possible to the power and ground pins.

2) **Ground Bounce:** The voltage distribution is maximum and minimum at certain points on the plane. This non uniformity or fluctuations are called Plane Bounce or Ground Bounce. This results in Resonance (very high impedance) and Anti resonance (very low impedance) points. These effects can be seen by a Full Wave 3D simulator. Impedance profile of power planes is also used to visualize these effects. Decoupling capacitors are used to nullify these effects. The impedance of power planes should be minimized as much as possible [32]. To reduce the impact of return current switching noise it is better to time scatter the source of ground plane noise.

**C. Bit Error Rate**

In high speed devices and systems, Bit Error Rate or Bit Error Ratio (BER) is the most important factor to characterize the system. BER is calculated by taking the ratio of number of bits misinterpreted to the total number of bits sent or transmitted. In USB 2.0 system, BER of \( 10^{-12} \) is acceptable but to achieve a good confidence level, it should be \( 10^{-15} \) [3]. BER is mainly due to two reasons, Time Jitter and Amplitude Noise.

1) **Time Jitter:** Jitter is temporal variation in the signal. As shown in Figure 1, rising and falling edges deviate from their ideal position. There are three major sources for jitter clock signal generation, power supply variation and coupling capacitance [10]. Jitter is mainly of two types - “Deterministic Jitter (DJ)” and “Random Jitter (RJ)”. DJ can further be classified in to Data Dependent Jitter (DDJ) and Periodic Jitter (PJ). DDJ can further be classified in to Duty Cycle Distortion (DCD) and Inter symbol Interference (ISI). DJ is generally predictable. RJ is unbounded and its distribution meets the “Gaussian Function”. Its value is represented in terms of Unit Interval (UI) at some BER e.g. 0.1 peak-to-peak RJ at \( 10^{-12} \). PJ is the component of TJ that repeats after a certain frequency interval. This is generally characterized by sinusoidal. DCD is the variation in the edge of transition from its allocated time. Tailfit Algorithms are used to separate RJ from the measured distribution [27].

Total jitter (TJ) is given by

\[
TJ = RJ \ast DJ
\]

where \( \ast \) represents convolution, \( T_b \) is bit period and \( E_o \) is eye opening. TJ can only be measured by BERTs (Bit Error Ratio Testers). If the clock, which we are using as a reference for jitter measurement, is recovered from data itself, then jitter is no problem for us. Jitter is a major problem in SI so Jitter Attenuation Circuits (JAC) are used to reduce the jitter [11]. For accurate jitter measurement, various steps should be followed [59].

![Fig. 1. Jitter](image)

2) **Amplitude Noise:** Amplitude Noise may cause the misinterpretation at the receiving end. A ‘1’ can be interpreted as ‘0’ or vice versa due to noise, as shown in Figure 2. There are a lot of sources of noise. The important types of noise are reflection noise, coupling noise, ground bounce noise, intrinsic noise and supply noise [5], [6]. Many of the amplifiers add noise in the signal due to silicon level agitation. This amplification noise is also frequency dependent. In PHY, signal is amplified at receiver and this is also one of the causes of addition of noise. There are generally two types of amplitude noise, Random Noise and Deterministic Noise.

![Fig. 2. Amplitude noise](image)

**D. Signal and Power Delivery**

In the context of High Speed Systems, Signal and Power Delivery Networks are described in this section. SI is generally associated with Signal Delivery Network(SDN) and PI is associated with Power Delivery Network(PDN), but they cannot be separable. There are two paths in a System, one is Signal Path and the other one is Power Path. Power is supplied by VRM through board and packages to the core circuit. High speed signal (high speed transceiver o/p in the case of serial links), on the other hand, is generated from core circuit and is transmitted to board through wire bonds, package nets etc. These two paths are shown in Figure 3. However the direction of both the paths are opposite to each other, they interfere [11].
1) Power Path: Power distribution in the system is maintained by PDN. In literature, there are methods introduced to design a system considering power network and signal network separately which cannot predict interaction between power and signal. Power path starts from the regulator on the board, then continues from the power planes of the board to the power planes of the package and last, from the package to the chip. There is again a systematic network for on-chip power delivery network. Figure 3 shows power path. (Power path and signal path are not the standard terminologies but these have been introduced here for the easier understanding for co-analysis of PI and SI).

2) Power-Ground Planes in Package and Board: Power planes are used in multilayered packages and boards for PDN. Power planes in PDN, act as cavity resonators. These planes provide high impedance path to the return displacement current at resonant frequencies when there are interconnect discontinuities within the system [42]. For power distribution in a board or a package, there are mainly two approaches [57], 1. Power Grid based PDN and, 2. Reference Plane based PDN. Shielding is better in grid based PDN but still, for the simplicity of the analysis of inductive and capacitive effects, Reference Plane based PDN is used enormously. The later one also reduces the bus line self inductance and it provides zero return path resistance at lower frequencies as well [28]. In our system, reference plane based design is used. Figure 4 shows the s-parameters of the plane used in this analysis. In multi layered packages and boards, in addition to the coupling in the transverse direction, there may be coupling in vertical direction from one plane to the another. Also, the transmission lines may contribute to the return path discontinuities like vias, half planes etc. [31]. Since the size of packages is very large so it is very difficult and time consuming to analyze packages using full wave simulators. Large packages consist thousands of signal lines and these signal lines are placed in many layers. The signal lines are placed between power/gnd planes to have an “impedance-controlled” board or package. A power/gnd plane minimizes the coupling between two signal lines so there are many gnd/planes included in the board. Planes are capacitive at low frequencies while inductive at higher frequencies [49]. Due to the low inductance provided by planes, these are used to supply the charge to the switching circuits through decoupling capacitors.

In vertical direction, there is a lot of possibility that the noise coupling occurs at the edges of the planes so the accurate modeling is a critical job. The same issue takes place at vias also. At vias, the mode changes due to the return path discontinuities [31]. Finite Difference Modeling (FDM) is a very important efficient modeling scheme for multi layered plane/gnd structures [33]. However [30] provides a good method to extract equivalent circuit for power/gnd planes but it is restricted to one pair of planes only. Package planes are useful for power distribution in mid frequency range (1 MHz to 100 MHz) to high frequency range (above 100 MHz). At resonance frequency, this becomes the significant source of noise in package. The resonance frequency of the planes is given by

\[ f_{mn} = \frac{1}{2\pi} \sqrt{\mu\epsilon} \cdot \sqrt{(m\pi/a)^2 + (n\pi/b)^2} \]  

where \( m \) and \( n \) are the mode parameters.

3) PDN in Die: An On-chip PDN contains,

1) Power Rings
2) Power Straps
3) Power Trunks
4) Instances

An approach for modeling and analysis of On-Chip PDN is given in [37]. As current required by die is increasing continuously and voltage margin is also decreasing continuously, there should be low impedance path between power supply source and die to fulfill the power requirements [45]. There must be very high impedance path between power plane and signal plane. Typically, 5% of the power-ground voltage difference is the upper limit for the noise to maintain Power Integrity [49], [50], [51]. The current requirements by ICs has been increased up to 25 A, according to [51] and 100 A, according to [49]. There are various good papers on on-chip PDN [47].

4) Decoupling Network: There are various decoupling capacitors at various stages of system e.g. discrete decoupling capacitor on PCB between ground and power plane of PCB [45]. Decoupling capacitors are distributed over package, board and ICs, to store the charge and supply whenever needed by IC [49]. Embedded passive components have low parasitic, i.e. low ESL (Equivalent Series Inductor) and ESR (Equivalent Series Resistance) parameters [53], Surface Mount Decoupling (SMD) capacitors are ineffective at frequencies above 100 MHz due to the large inductance provided by these capacitors so embedded Capacitors are used [53].

5) Signal Path: Signal travel through the interconnect media in one of possible options like microstrip line, stripline or traverses between layer by vias.
6) Microstrip Lines: This transmission line configuration include a central conductor running over a reference ground plane with dielectric layer in between them. This configuration is mostly used on FR4 material to provide the cost economic solution in general. Sufficient literature is available to calculate the characteristic impedance of this transmission line.

7) Strip Lines: This is another form of transmission line wherein the central conductor is sandwiched between two planes. This transmission line attenuation is very much dependent on the quality of dielectric constant. Closed form equations are available in literature to calculate the characteristic impedance of this transmission line.

8) Vias: Signal propagates between different layer through Vias. Figure 5 shows typical vias in a package net. Via can be considered as a π network [34]. Via introduces delay in signal and the capacitance of the via slows down the edge rate which results in jitter. There is significant degradation in Signal Quality while signal travels on traces just above the split planes, due to voids in power/gnd planes [60]. Via inductance reduces the effect of decoupling network. Via inductance is given by [34]:

\[ L_{\text{via}} \approx 5.08 \cdot h \cdot \ln \left( \frac{4h}{d} \right) + 1 \\ \text{nH} \]  

where \( h \) is barrel height and \( d \) is barrel diameter.

9) Wires Bonds: Wire Bonds: Wire Bonds are used to communicate between a Die and the outside world. Wire bonds are connected to the I/O rings. Various methods are available for modeling of wire bonds [35] [43]. Wire bonds behave as Low Pass Filters (LPF) [44]. The main independent factors in the case of wire bonds are - (a) maximum loop height, (b) die height and (c) span loop.

\[
\text{Fig. 5. Wire bonds: side view and top view [13]}
\]

III. SI-PI AS A DUAL

There are two paths in a system, as described in section 2.4. However the direction of both the paths are opposite to each other, they interfere [11]. This interference can be understood as ‘interaction’ in terms of statistics. This is discussed in more detail in our previous paper [36]. In earlier literature, there are methods introduced to design a system, considering power network and signal network separately, this cannot predict interaction between power and signal lines. There are some new research papers like [24] [48] [53] [38] for co-design of SI and PI. In SI-PI Co-Design, three main factors to be considered are [24]:

1) Noise sources,

2) Noise coupling path between Power Plane and Signal Traces,

3) Resonance on Signal Traces.

We have considered PDN with SPN in our analysis as it includes coupling between power plane and signal traces. This coupling will affects both PDN and SPN performance.

IV. SERIAL LINKS

A. Introduction

Serial Links are the links in digital and mixed signal systems through which the data transmits serially from one system to another. The most popular serial links are USB, MIPI, HDMI, SATA etc. Serial Links are becoming more significant at higher data rates; as the problems of the skew and high pin-count are very much reduced. But, on the other side, the links that have the clock signal embedded in the transmitted data, have some challenges for transmission quality [21].

B. Need for Serial Links

At high speed, the chip-to-chip connections, inside high speed digital systems like routers, servers etc., have undergone the architectural shift from parallel buses to serializer / deserializer (SERDES) links. The main advantage of using serial links is that they don’t cause data skew problems [6]. For transmission purposes, serial links require less number of wires compared to parallel links [41].

C. Eye Diagrams

Eye Diagrams can be visualized by overlapping a hundreds of bit in one unit interval (UI). Figure 6 shows a typical standard eye diagram (of netlist only) with Mask and Figure 18 shows the same with degradation of bit stream after transmission through channel. A Mask (pink colour in Figure 6) defines upper and lower limits of amplitude and jitter. Mask can be defined according to the application in which we are using the system. This eye is simulated in Agilent ADS. There are a lot of others tools dedicated for Eye Simulation. Sometimes, simulations are impractical in time domain so statistical processing is used [18]. StatEye is one such tool.

\[
\text{Fig. 6. A 250 nsec bitstream and corresponding eye diagram}
\]
D. Differential Signaling

Differential signaling technique reduces common mode noise. Differential signaling is widely used in serial links. In differential signaling, per unit length (p.u.l.) inductance decreases with increase in frequency [56]. Differential signals are also useful to represent the “Squelch State” of serial links. This state means that the signal at its connector is not valid and the link is called in squelch state [3].

V. SYSTEM MODELING

A. Introduction: Complete System

There are two paths in a system, as described in section 2.4. These two paths are shown in Figure 3. However the direction of both the paths are opposite to each other, they interfere [11]. While designing a system, for SI, we consider an ideal power delivery network (PDN) which is not possible practically. In older literature, there are methods introduced to design a system having been considered power network and signal network separately which cannot predict interaction between power and signal. So we are using SI and PI as a dual (as discussed in section 2.5).

The system is analyzed for USB 2.0, it consists of die, package, board and its measurement has been done by high bandwidth oscilloscope. Squidd card and cable were used for measurements of eye diagram. To include the measurement environment effect, their S-parameters are measured by Time Domain Reflectometer (TDR). Transceiver design netlist is used in simulations. Package is designed using Cadence SiPR and board is designed using Allegro (cadence). System’s equivalent model is simulated in Agilent ADSR, as shown in Figure 17.

1) Die: Die includes both core circuit and I/O circuit. Core circuit is the circuit that communicates within the die while I/O circuit communicates outside the die i.e. from the external world. Die can be included in complete system analysis either by including its design files or its response to a bit stream stimulus. In case complete design files is used in the simulation model, it will be much slower and the simulation will be very inefficient in terms of time and cost. Hence V/T curves are used. A PRBS (Pseudo Random Bit Stream) has been taken in this analysis. Worst case bit patterns give best results [65]. Figure 6 shows the output bit stream and eye diagram for USB 2.0 transceiver in its high speed mode, when stimulated by a PRBS. Some alternate models for including die in the system level analysis, are also introduced now a days. One of them is Chip Power Model (CPM) [8]. This provides the output current profile of Differential line which can be used later as to insert the effects of chip for power delivery network design.

2) Package: A package provides electrical connections as well as mechanical support to the chip (die). Chip is encapsulated in a package. A package provides electrical connections to the I/O circuit of the chip. There are various types of packages like Ball Grid Array (BGA), Flip Chip, Pin Grid Array (PGA) etc. In our analysis, the package used is BGA. Figure 8 shows routing of one net in a BGA package; while Figure 9 and Figure 10 show the physical view and crosssectional view of a BGA package respectively.

SiP stands for System in Package. SiP is a package that contains all the components or subsystems that are needed to build a complete system e.g. a package that contains Digital IC, Analog IC, RFIC, Passive Components etc. [46]. There may multiple dies, vertically or horizontally. Figure 8 shows routing of one net in a BGA package; while Figure 9 and Figure 10 show the physical view and crosssectional view of a BGA package respectively.

3) Wire Bonds: Bond wires connect die to package substrate. Fig. 5 shows photograph of wire bonds connecting die to substrate. A wire bond profile report was generated from Cadence SiPR, and the information from the same was fed to the existing elements in ADS. Wire bonds on the left side show the differential signals coming from the die while wire bonds at the right side show power supply wires to the chip.
4) **Board:** A bareboard is the board that contains no components mounted on it (Figure 14). For power delivery network analysis, one should check for the resonance/antiresonance frequency bands. At resonance frequency, there is one point on the board that has low impedance between power and ground planes. There are various numerical methods to find out the impedance profile of a bareboard [55]. At anti resonance frequency bands, PDN offers max impedance. According to this, we can choose the exact point where to place capacitors to nullify the effect of resonance/anti resonance and the value of the capacitor for the same [50].

5) **VRM:** VRM (Voltage Regulator Module). SPICE files (equivalent RLC model) of VRM are provided by the vendors to show customers the performance of their IP. We can use this spice file for simulation purposes, to realize the behavior of VRM. The standard 3.3 V VRM from National Instruments is used.

6) **Decoupling Network:** Decoupling Network consists of capacitors. The capacitors used in the USB 2.0 testchip system were provided by Murata Manufacturing co. Ltd. The s-parameter files are provided by the vendors to simulate the effects of decoupling network. S-parameters files are included to realizing the behavior of decoupling network.

**B. Including Measurement Effects**

In measurements, we use cables and squidd card. In S-parameters measurements of a passive network or a subsystem, we consider the subsystem as a black-box and we concentrate on the input-output characteristics of the subsystem [52]. S-Parameters describe a component as a black box as well as map the behavior of component over a range of frequencies. S-parameters are saved in touchstone file format. Touchstone files are efficient only for small signal models. Figure 15 shows typical s-Parameter plots for a standard Squidd card and a USB cable combined. Characteristics impedance of 50 ohm is the standard Zo to maintain an optimized power handling capacity and attenuation and the tradeoff between them [54]. The input impedance of a transmission line may be time dependent [61].

**C. Model to Hardware Correlation**

Here the analysis of a USB 2.0 system is shown. Figure 16 shows the model used for SI-PI co-analysis to correlate with the hardware. This is described in detail in our previous paper [66]. This is more accurate (Figure 18) in comparison with the conventional models which considers only SI for eye diagram analysis. This model was designed in Agilent ADS tool. Figure 17 shows USB 2.0 complete system implementation in ADS.

**VI. QUALITY ENGINEERING**

An electrical engineer Genichi Taguchi worked on Statistical Process Control and Factorial Design of Experiments
DOE) and introduced some new methods called “Taguchi Methods” [22]. That were responsible for the boom in the Japanese Industry after second world war. Taguchi Methods are the backbone of the “Quality Engineering”, a branch of engineering that deals with the yield and the quality of the production. NASA and Department of Defence, USA, jointly initiated a revolutionary management strategy for quality management and cost reduction called “Total Quality Management (TQM)”. TQM is totally based on Taguchi Methods [23]. According to Taguchi ‘Quality’ is defined as, “The quality of a product is the minimum loss imparted by the product to the society from the time product is shipped” [23]. In the paper, DOE techniques are used for robust optimization of USB 2.0 System. For Orthogonal Experiments, Taguchi Methods are used to optimize or diagnose the system. This section explains basic theory of DOE and robust optimization.

A. Design of Experiments (DOE)

Design of Experiments (DOE) is an important method to efficiently design a system maintaining its important output parameters within the robustness limits of the functionality of the system. DOE is used to ensure the value of the selected output parameter (which is called Quality Parameters) within defined range, when the system has unwanted and uncontrollable design variations. Thus, DOE is a method to design a system in a robust way as well as meeting the system output requirements. A basic example for DOE is manufacturing of an IC. An IC may get affected by temperature, humidity, power fluctuations or many other variations. These are uncontrollable variations but the design should work properly in varying conditions also. In most of the production development process, DOE is used because it is more effective when simulation and verification are difficult to achieve the desired results. According to Shina [2],

“Design of Experiments is a systematic method for determining the effect of factors and their possible
interactions in a design or a process towards achieving a particular output of the quality characteristic(s). The ‘treatments’ are the well defined procedures or experiments to examine a system for its output characteristics. For example, if someone, while observing the output response of a circuit, uses different waveforms like sawtooth, sine, rectangular etc., these are called different treatments. An ‘interaction’ is the variation among the differences between means for different levels of one factor over different levels of the other factor.

1) Full Factorial Experiments: Full Factorial Experiments or Complete Factorial Experiments consist of all possible combinations of the individual factors at their all possible levels of variation [9]. Complete factorial experiments become very large number when the variables are more e.g. if we have 13 random variables each with 3 levels in our system (this is the case of the paper itself), we’ll have to perform $3^{13} = 15,94,323$ experiments, which is not feasible.

However full factorial experiments provide more accurate results in robust optimization, these have limitations for larger systems. Full factorial experiments are more useful rather than performing experiments to investigate the effect of single factor. Reason for the same is that in Full Factorial Experiments the ‘interaction’ among factors can be taken in to account. [9].

2) Fractional Factorials: Fractional Factorials are the experimental test programs, by which we can find the validity of a system using only a small fraction of all the test conditions. As the total number of treatment combinations may be very large, we intend to test some of these combinations, giving a fractional factorial design. The set of experiments can be defined either by an orthogonal array, or by forming a subgroup of the direct product of abelian groups of orders equal to the number of levels of each factor (in our case this is 3).

B. Statistical Analysis of DOEs

A method called Analysis of Variance (ANOVA) is used for statistical analysis of DOEs. ANOVA determines the significance of each factor as how much it affects the output quality characteristics. Following are the general formulae (stated in [2]) how to find out the contribution of each factor or random variable, in the output quality characteristic.

Total sum of the squares is

$$SS_T = \sum (Y_i - Y_{average})^2 = \sum Y_i^2 - \frac{\sum Y^2}{n} \quad (7)$$

Also, sum of Squares for each factor is

$$SS_F = \left( \frac{\sum_{n_{level1}} Y_{level1}^2}{n_{level1}} \right) + \left( \frac{\sum_{n_{level2}} Y_{level2}^2}{n_{level2}} \right) + ... - \left( \frac{\sum Y_i^2}{n} \right) \quad (8)$$

Examples of Degrees of Freedom (DoF)are:

1) DoF Orthogonal Array = Number of experiments - 1
2) DoF Factor = Number of levels - 1
3) DoF Interaction = Product of the DoF of each factor

4) DoF Error = Total DoF - DoF of significant factors and interactions
5) Variance $V = SS/DoF$ (This is also called Mean Square Deviation (MSD))

Standard deviation of total experiments:

$$V_T = \left( \sigma_{total experiment} \right)^2 \quad (9)$$

$F$ Ratio for each factor:

$$F_e = \frac{V_F}{V_{error}} \quad (10)$$

Modified sum of squares for each factor:

$$\left(SS_F'\right) = SS_F - V_{error}.DoF_F \quad (11)$$

Percentage contribution:

$$(p\%) = \frac{SS_F'}{SS_T} \quad (12)$$

where $(p\%)$ shows how much an independent factor contributes in to the total quality parameter output.

C. Robust Optimization

Robust optimization means to optimize a design in such a way that it will certainly work according to the specifications for which it is desired to work. Taguchi Methods are used for general Robust Optimization. There are various other methods also that perform Risk Analysis for robust optimization [26]. The approach used in the paper is to make the robust optimization cost effective which is described thoroughly in our previous publication (readers may refer it for more details) [40].

VII. STATISTICAL ANALYSIS

Worst case analysis of a system may underestimate a system, so statistical analysis is required. Worst case has very low probability of occurrence [20]. The objective of “Statistical Modeling” is to reduce the number of parameters of the model, to the smallest possible uncorrelated set, in such a way that this set can generate all the parameters when needed [19]. Monte Carlo Simulation is useful when there is a strong and complex correlation between the parameters of a device. This is the case in IC manufacturing because the devices are manufactured simultaneously but at system level, since all the components are manufactured separately, Monte Carlo simulation is not very useful [17]. The objective of using statistical methods in this paper is to analyze at System Level. We want the best information in the least possible simulations. The system consists of different components so we will not use Monte Carlo simulation here. In the paper, Fractional Factorial experiments are used and the set of experiments are according to Taguchi’s array $L_{27}(3^{13})$.

A. Quality Parameters and Factors

The Quality Parameters used for the analysis are time jitter and eye amplitude. The factors affecting them are chosen by the design. The focus while choosing the factors was that what were the factors that might affect the system and that can be changed while simulating the design. And in addition,
the factors chosen are such that they can be modified by the designer for the optimized solution. Here, w2 is the only factor that is taken as uncontrollable factor with large deviation, but that can also be redesigned. The selection of factors was 3. According to this the 13 factors are:

1) Board power plane metal thickness (b1)
2) Board power plane substrate thickness (b2)
3) Board power plane dielectric constant (b3)
4) Board signal trace width (b4)
5) Board signal trace length (b5)
6) Board traces differential pair spacing (b6)
7) Package via diameter (p1)
8) Package signal net width (p2)
9) Package signal net differential gap (p3)
10) Package dielectric constant (p4)
11) Diameter of crossection of wire bonds (w1)
12) Angle of wire bonds (w2)
13) Termination at receiver (t)

Table I shows the values of mean and deviation of the 13 design factors.

### B. Taguchi Arrays and Simulations

In our case the random variables or factors are 13 and the levels are 3, so the Taguchi Array of $L_{27}(3^{13})$ is used. The experiments (also called as treatments) or the orthogonal simulations performed are planned according to this array. This is shown in Table II. The same case study is presented in more details in [39]. Here -3 represents $\mu - 3\sigma$, 0 represents $\mu$ and 3 represents $\mu + 3\sigma$ for a design parameter. Results of the corresponding simulations are shown in Table III. There are various outputs shown in the Table III which are the parameters of an eye diagram. These are Eye Amplitude, Eye Height, Eye Opening, SNR, Jitter etc. We are taking Eye Amplitude and Jitter (peak to peak) as our output quality parameters. The $L_{27}(3^{13})$ Taguchi Array is used from [1].

### C. Statistical Analysis

Based on the formulae given in section 5.4, we can find the contribution by each factor. Based on this we can formulate it and can optimize it. To calculate the percentage contribution $p\%$ for each factor will be very exhaustive. Alternately we can use the matrix methods used in literature [17]. Following are the matrix method equations (6.1, 6.2) used to find the sensitivity coefficients for each parameter. $\alpha = \{\alpha_0, \alpha_1, ..., \alpha_{13}\}$ and $\beta = \{\beta_0, \beta_1, ..., \beta_{13}\}$ are the vectors containing sensitivity coefficients of b1, b2, ..., w1, w2, t for Jitter and Eye Amplitude respectively. $[A]_{27 \times 13}$ is the experimental matrix. Since Jitter and Eye Amplitude are affected by some other parameters also which are not taken in to account for the experiments or ‘Treatments’ performed in the paper. These factors include both controllable and uncontrollable factors. The effect by them is called ‘error’. Error vectors are used to represent this. $[e_1]_{27 \times 1}$ and $[e_2]_{27 \times 1}$ are the Error Matrices. $Cov(J_r, E_a)$ is Covariance between Jitter and Eye Amplitude which shows how these Quality Parameters vary simultaneously. $Cov(J_r, E_a)$ and $Cov(E_a, E_a)$ are simply the variances of Jitter and eye Amplitude respectively. $R^2$ is the regression coefficient, which tells about the Model Fitness i.e. how much the model takes in to account the quality parameter [58].

$$\alpha = ([A]^T A)^{-1} [A]^T [J_r]$$ (13)

$$\beta = ([A]^T A)^{-1} [A]^T [E_a]$$ (14)

The values of vectors $\alpha$ and $\beta$ are given in Table IV.

$$[e_1]_{27 \times 1} = J_r - [A]([A]^T A)^{-1} [A]^T [J_r]$$ (15)

$$[e_2]_{27 \times 1} = E_a - [A]([A]^T A)^{-1} [A]^T [E_a]$$ (16)

$$Cov(J_r, E_a) = E[(J_r - \mu_J)(E_a - \mu_E)]$$ (17)
The simpler method to find the Covariance is

\[ \text{Cov}(J_r, E_a) = \sum_{i=1}^{8} \alpha_i \beta_i \sigma_x^2 = 0.0376 \]  

(18)

This small value indicates that both the quality parameters don't change in a strong manner i.e. if jitter changes, the probability of changing eye amplitude is very less.

\[ \text{Cov}(J_r, J_r) = \sum_{i=1}^{8} \alpha_i \sigma_x^2 = 293.9799 \]  

(19)

\[ \text{Cov}(E_a, E_a) = \sum_{i=1}^{8} \beta_i \sigma_x^2 = 0.00048 \]  

(20)

Regression Coefficients are calculated as :

\[ R^2_{J_r} = 1 - \frac{\sum_{i=1}^{27} [e_i]^2}{\sum_{i=1}^{27} [J_r - \bar{J}_r]^2} = 50.20\% \]  

(21)

\[ R^2_{E_a} = 1 - \frac{\sum_{i=1}^{27} [e_i]^2}{\sum_{i=1}^{27} [E_a - \bar{E}_a]^2} \approx 47\% \]  

(22)

The above values of \( R^2 \) indicate that some of the parameters are not taken into account for variation. This is mainly due to PLL jitter in die, which is ignored. For all the simulations, the netlist included is USB 2.0 high speed transceiver netlist which considers an ideal clock source. While taking PLL into account, the time for simulation becomes very long (in days). For such a statistical analysis (which has been used in the paper), which requires a lot of simulations, this is not feasible including PLL into system.

1) Sensitivity Functions: Since all the design parameters are having different units of measurement i.e. \( \mu m, inch, \Omega \) etc., thus to formulate the Quality Parameters as the function of various design parameters, it is necessary to normalize all the design parameters. To normalize a design parameter \( x \), the process is:

\[ X_{\text{norm}} = \{ y | y = \frac{x_i - \mu_x}{\sigma_x}; \forall x_i \in X \} \]  

(23)

After normalization of all the design parameters and finding their sensitivity coefficients, we can find the expression of Quality Parameters as linear function of design parameters. Based on the values from above equations (and given in Table IV) the formulation for Jitter and Amplitude noise are:

\[ J_r = \alpha_0 + \alpha_1 b_1 + \alpha_2 b_2 + \alpha_3 b_3 + \alpha_4 b_4 + \alpha_5 b_5 + \alpha_6 b_6 + \alpha_7 p_1 + \alpha_8 p_2 + \alpha_9 p_3 + \alpha_{10} p_4 + \alpha_{11} t + \alpha_{12} w_1 + \alpha_{13} w_2 + \epsilon_1 \]

and

\[ E_a = \beta_0 + \beta_1 b_1 + \beta_2 b_2 + \beta_3 b_3 + \beta_4 b_4 + \beta_5 b_5 + \beta_6 b_6 + \beta_7 p_1 + \beta_8 p_2 + \beta_9 p_3 + \beta_{10} p_4 + \beta_{11} t + \beta_{12} w_1 + \beta_{13} w_2 + \epsilon_2 \]

In the above expressions of \( J_r \) and \( E_a, \epsilon_1 \) and \( \epsilon_2 \) are model errors. This is how Jitter and Eye Amplitude vary according to the design parameters. One thing to be noted here is that the first term in both the equations are the mean (\( \mu \)) values of \( J_r \) and \( E_a \). If we have the expression without \( \alpha_0 \) and \( \beta_0 \), the mean values of the expressions will be 0. This is so because all the design parameters are normalized in the equations.

### TABLE IV: SENSITIVITY COEFFICIENTS FOR JITTER AND EYE AMPLITUDE

<table>
<thead>
<tr>
<th>S.No</th>
<th>Sensitivity Coefficients for Time Jitter</th>
<th>Sensitivity Coefficients for Eye Amplitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \alpha_0 = 160.9046 )</td>
<td>( \beta_0 = 0.3879 )</td>
</tr>
<tr>
<td>2</td>
<td>( \alpha_1 = 2.5178 )</td>
<td>( \beta_1 = 0.0000 )</td>
</tr>
<tr>
<td>3</td>
<td>( \alpha_2 = 0.7102 )</td>
<td>( \beta_2 = 0.0015 )</td>
</tr>
<tr>
<td>4</td>
<td>( \alpha_3 = 0.3542 )</td>
<td>( \beta_3 = 0.0002 )</td>
</tr>
<tr>
<td>5</td>
<td>( \alpha_4 = 3.4812 )</td>
<td>( \beta_4 = 0.0010 )</td>
</tr>
<tr>
<td>6</td>
<td>( \alpha_5 = 2.8084 )</td>
<td>( \beta_5 = 0.0009 )</td>
</tr>
<tr>
<td>7</td>
<td>( \alpha_6 = 8.2215 )</td>
<td>( \beta_6 = 0.0056 )</td>
</tr>
<tr>
<td>8</td>
<td>( \alpha_7 = 3.2486 )</td>
<td>( \beta_7 = 0.0029 )</td>
</tr>
<tr>
<td>9</td>
<td>( \alpha_8 = 12.4146 )</td>
<td>( \beta_8 = -0.0022 )</td>
</tr>
<tr>
<td>10</td>
<td>( \alpha_9 = 5.9403 )</td>
<td>( \beta_9 = 0.0036 )</td>
</tr>
<tr>
<td>11</td>
<td>( \alpha_{10} = 0.7007 )</td>
<td>( \beta_{10} = 0.0039 )</td>
</tr>
<tr>
<td>12</td>
<td>( \alpha_{11} = 10.4356 )</td>
<td>( \beta_{11} = 0.0061 )</td>
</tr>
<tr>
<td>13</td>
<td>( \alpha_{12} = 8.1475 )</td>
<td>( \beta_{12} = 0.0007 )</td>
</tr>
<tr>
<td>14</td>
<td>( \alpha_{13} = 8.9280 )</td>
<td>( \beta_{13} = 0.0007 )</td>
</tr>
</tbody>
</table>

### D. Optimization

In Figure 20, there is shown the independent effect of each parameter on Jitter and Eye Amplitude. This is calculated by averaging the response at each level [18]. We can see that some of the parameters at some of the levels, provide better results. b5 is better at -3 level for both Jitter and Eye Amplitude than 0 and +3 levels, so the obvious choice for the optimized design will be -3. If we want to optimize the design for Jitter only, then we will consider the levels that provides least jitter like...
TABLE V
COMPARISON OF STANDARD DESIGN AND OPTIMIZED DESIGN AS SHOWN IN FIGURE 19

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Standard Design</th>
<th>Optimized Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eye Amplitude</td>
<td>0.6388 V</td>
<td>0.6492 V</td>
</tr>
<tr>
<td>Eye Width</td>
<td>1.8492 nsec</td>
<td>1.9729 nsec</td>
</tr>
<tr>
<td>Jitter</td>
<td>139 psec</td>
<td>111 psec</td>
</tr>
</tbody>
</table>

0 for $b1$ and -3 for $b4$. The same case is applicable for eye Amplitude also but the levels should be chosen in such a way that the Amplitude becomes greater. This was the case with a single Quality Parameter. But if we want to optimize the design for both Jitter and Eye Amplitude, in the first first the levels that provide least jitter and best eye amplitude should be identified. These are -3 level for $b5$, +3 for $p1$, -3 for $p2$, -3 for $p3$ (jitter is approximately equal for -3 and 0), -3 for $w^2$ and 0 for $t$. The second step is to find the percentage change in both the quality parameters at -3 and +3 levels from the quality parameter with the standard design i.e. design with all parameters set to $\mu$. If the second step is not followed, the design will still be optimized. Following the first step, the optimized design in this case is:

- $b5 \rightarrow -3$
- $p1 \rightarrow +3$
- $p2 \rightarrow -3$
- $p3 \rightarrow -3$
- $w^2 \rightarrow -3$

All the other parameters will be at 0 or $\mu$. Fig. 19 shows the eye diagram for optimized design for USB 2.0 testchip system. We can see that in comparison with standard design (with all parameters set to $\mu$), jitter is about 20% lesser and eye amplitude is almost 2% higher. Comparison between both the designs is shown by Table V.

Fig. 19. Eye diagrams for standard design and optimized design (refer to Table V for values of parameters)

VIII. REFLECTION GAIN ENHANCEMENT

A. Concept of Reflection Gain

Now a days, a new concept or Reflection Gain is introduced [32] [12]. At the termination, instead of matched termination, a high impedance element is introduced. This mismatch in impedance values, reflects the signal back. The amount of reflection depends upon the characteristics of this high impedance line. If the line is purely inductive, it provides an overshoot.

B. Circuit for Differential Signaling

Reflection gain analysis for a very simple system which has one transmitter, one receiver, both connected through a transmission line is shown in [32]. In our case of USB 2.0 system, we are having differential lines as output. In the case of our system, it is USB 2.0 differential signaling. A 2-port circuit used for this purpose is shown in fig. 21 where $L_1 = L_2 = L$.

Fig. 21. 2-Port circuit for reflection gain

C. Frequency Domain Analysis of Various Channels

We can also analyze a channel in frequency domain. Transfer Function of various channels are shown in fig. 22. By varying the value of $L$, we test various conditions. We find $8nH$ is a suitable value for us. This can be visualized by fig. 22 also. In Figure. 22, there is shown results for various test experiments of Table VI. We can see that insertion loss is minimum in test3.
TABLE VI

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Value of L</th>
<th>Eye Amplitude</th>
<th>Jitter Width</th>
<th>Eye SNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>1 nH</td>
<td>0.6475</td>
<td>74</td>
<td>2.0230</td>
</tr>
<tr>
<td>Test 2</td>
<td>4 nH</td>
<td>0.6511</td>
<td>74</td>
<td>2.0142</td>
</tr>
<tr>
<td>Test 3</td>
<td>8 nH</td>
<td>0.6548</td>
<td>74</td>
<td>2.0138</td>
</tr>
<tr>
<td>Test 4</td>
<td>10 nH</td>
<td>0.6542</td>
<td>74</td>
<td>2.0137</td>
</tr>
<tr>
<td>Test 5</td>
<td>12 nH</td>
<td>0.6579</td>
<td>74</td>
<td>2.0142</td>
</tr>
<tr>
<td>Test 6</td>
<td>15 nH</td>
<td>0.6602</td>
<td>86</td>
<td>2.0132</td>
</tr>
<tr>
<td>Test 7</td>
<td>20 nH</td>
<td>0.6698</td>
<td>80</td>
<td>2.0128</td>
</tr>
<tr>
<td>Test 8</td>
<td>25 nH</td>
<td>0.6724</td>
<td>82</td>
<td>2.0119</td>
</tr>
<tr>
<td>Test 9</td>
<td>0.1 nH</td>
<td>0.6332</td>
<td>72</td>
<td>2.0390</td>
</tr>
<tr>
<td>Test 10</td>
<td>0.2 nH</td>
<td>0.6344</td>
<td>72</td>
<td>2.0312</td>
</tr>
<tr>
<td>Test 11</td>
<td>0.4 nH</td>
<td>0.6389</td>
<td>72</td>
<td>2.0287</td>
</tr>
<tr>
<td>Test 12</td>
<td>0.8 nH</td>
<td>0.6414</td>
<td>72</td>
<td>2.0242</td>
</tr>
</tbody>
</table>

D. Modified USB 2.0 System

USB 2.0 Testchip system design has been modified by robust optimization followed by reflection gain addition. Eye Amplitude is 0.6548 and Jitter is 74. Compared to the standard design (Figure 18), there is almost 50% improvement in Jitter and about 2.5% improvement in Eye Amplitude. Corresponding Eye Diagrams results are shown in Figure 23, which shows two eye diagrams. First eye diagram is of standard design while second one is of finally modified design.

Fig. 22. Insertion loss of various channels

Fig. 23. Improvement in USB 2.0 testchip system performance (eye diagrams)

IX. CONCLUSION

Signal and power integrity methodology presented in the paper, which considers power path also, is more accurate than conventional models. But Path and Signal Path are not completely separable. They affect each other at higher speeds. Thus Signal integrity and Power Integrity must be taken together. Robust optimization is useful to improve system performance.

APPENDIX

A. Factors affecting Robustness

We can define robustness as “The quality of being able to withstand changes in procedures or circumstances” and Reliability as “The upper and lower bounds of design parameters of a system to maintain its functionality”. From simulations, we can find the robust limits (upper and lower bounds) of an IP, but some of the factors are not easy for simulations. However some of the tools provide the facility yet it is not effective.

1) Temperature: Analog circuits are much prone to temperature variations. Temperature variations cause leakage current and this is the dominating factor to reduce the efficiency of PHY transceiver. But we’ve not included temperature in this analysis, as our main objective is to analyze the channel rather than the die itself. Moreover, temperature affects channel in a very weak manner.

2) Process Variations: The same circuit fabricated in different processes have different signal integrity problems. A circuit design should be in such a way that its Process Sensitivity is lowest [10]. Thus robust process sensitivity of USB is very important factor to be determined.

B. Future Work

Research in Signal and Power Integrity has a very high potential. In the paper, co-analysis has been done for both SI and PI. This analysis can be highly improved if we consider PLL in to the system instead of ideal clock taken by transceiver. This will improve regression coefficient up to a very large value. The system analysis will be more meaningful if we do a mixed hierarchal analysis which takes in to account the variations in the channel as well as in the device itself.

Frequency domain analysis of channels may be done to improve the system performance. The transfer functions for various channels are shown in Figure 24. These are the transfer functions for the various values of reflection gain inductance L. We can see that some of the channels give spikes at some particular frequency. It means that channel when operable,
will not give good results in that frequency range. To avoid the undesired results, these spikes should be shifted in the frequency range which is not of our interest. This shifting can be done by adding some elements.

REFERENCES

Jai Narayan Tripathi was born in Gangapur (Bhilwara), a town in Rajasthan, India. He received his Bachelor of Engineering (Electronics and Communication Engineering) in 2007 and Master of Technology (Information and Communication Technology) from DA-IICT, Gandhinagar in 2009. The topic of his masters thesis was “Statistical Co-Analysis, robust optimization and Diagnosis of USB 2.0 System for Signal and Power Integrity”. He has worked as a Research Intern at Technology Research And Development Department (TR&D), STMicroelectronics Pvt. Ltd., Greater Noida, India for one year. He worked on Signal Integrity and Power Integrity issues for high speed serial links. At present he is a Ph.D Scholar at IIT Bombay, Mumbai. His areas of interest are Signal Integrity and RF Circuits Optimization. He is author of one book “Analysis of Signal Integrity and Power Integrity at System Level”. He has published in many international conferences and he serves as a reviewer for some journals like PIER, TEMC etc. He has 12 international research publication to his credit. Apart from academics, his hobbies are Creative Writing and Music. He has served as Music Director in an internationally screened movie and he has directed many dramas and his poems have been published in national magazines.

Rakesh Malik received his Bachelor of Engineering in Electronics and Electrical Communication from Punjab Engineering College, Chandigarh, India in 1986. He is Senior Group Manager in STMicroelectronics, India, leading Mixed Signal IP designs from specs to silicon in various technologies and applications. His present work is primarily in two domains. The design domain involves Analog and Mixed Signal IPs development like Analog to Digital, Digital to Analog Converters of various speed, precisions and various architectures. This has produced patents and publications in low power data converters applications for high speed ADC domain, OPAMPs architectures etc. The high speed serial link IPs developed caters to various speed and application includes links like USB, MIPI, ETHERNET PHY etc. In signal integrity and power integrity domain of high speed serial links, his work involves generic methodology development, system level robustness analysis of high speed serial links with complete analysis of board package and other interconnect media, modeling of power distribution network and decoupling network optimizations for high speed serial links. The work also involves statistical techniques, sensitivity analysis on parameter variations affecting serial link performance and optimization of channel performance considering the impact of signal and power delivery networks. He has many patents and publications on low area and low power architectures of FIR filters. His earlier career at Department of Electronics, India involves development around 32 bit microprocessor based systems architectures of FIR filters. His present work is primarily in two domains.