IT Infrastructure Architectures for Electric Utilities
- A Comparative Analysis of Description Techniques -

Jonas Andersson Pontus Johnson
jonasa@ics.kth.se pontusj@ics.kth.se
Dept. of Industrial Control Systems
KTH, Royal Institute of Technology
SE - 100 44 Stockholm, SWEDEN

Abstract
The escalating development of Information Technology enables utilities to reorganize or migrate from their existing disparate software systems towards an integrated Enterprise Software System Infrastructure (ESSI) that embraces the total organization. Integration of software systems is expected to increase competitiveness and to cut costs. However, since utilities’ present ESSI is heterogeneous as to type and technical platform, overlapping with regard to both data and functionality, and relying on ad-hoc low-level middleware, integration and management of ESSIs often turn out to be hazardous.

This paper presents a comparative analysis of the architectural modeling capabilities of established notations widely used by object-oriented, information engineering and structured methods, applied to typical problems found in utilities’ ESSI. Architectural descriptions may be used to visualize technical risks and opportunities in a utility’s current and future ESSI, as well as to improve communication between different groups within the utility, e.g. management and technical staff, and between the utility and its vendors of software systems.

1. Introduction
The discipline of software system architecture has traditionally been a domain for software developing organizations. However, as the process of managing the Enterprise Software System Infrastructure (ESSI) becomes more complex, the structure that architectural notations may provide grows relevant also for organizations that use software systems. Commonly, electric utilities use techniques such as business process modeling to elicit requirements on changes in its ESSI. This paper does not dispute that changes in an organization’s ESSI should be based upon its business goals. But, organizational changes and software systems often have very different life cycles. An ESSI must therefore be flexible enough to support both short and long-term organizational changes. Also, requirements based on organizational needs must be verified against constraints, risks, and opportunities in the organization’s present ESSI as well as against software available on the market. The process of designing an ESSI architecture should, unlike the acquisition of a single system, be considered as a design process – not a requirements elicitation process. Therefore, techniques that bring structure to the ESSI management process, taking both organizational and technical constraints and opportunities into consideration, are needed.

This paper presents a comparative analysis of architectural modeling capabilities of established notations used in structured, information engineering and object-oriented methods. In the paper, the notations are applied to typical configurations found in utilities’ ESSI. The ambition is to show that techniques originally intended for internal\(^1\) architectural modeling can also be used for external modeling of heterogeneous software system infrastructures in enterprises such as electric utilities. Furthermore, the paper stresses that a structured approach that makes use of well-defined notations for architectural modeling can improve communication between groups involved in the design of the ESSIs, such as top-management, end-users, system administrators, system owners, and vendors. The problems that are put forward have been identified during exploratory case studies of the ESSI management process in four Scandinavian utilities. The case studies have been carried out since 1997 by a research team from the Department of Industrial Control Systems at the Royal Institute of Technology (KTH), Stockholm, Sweden. The Scandinavian countries initiated the deregulation of the electricity market during the early and mid 1990s and therefore provide a rich source of experiences regarding...

\(^1\) The \textit{external perspective} on an ESSI, addressed in this paper, refers to the relationships between ESSI components, e.g. systems, databases, and connectors, whereas the \textit{internal perspective} addresses the decomposition of components in smaller parts such as modules.
the consequences of the deregulation on the electric utilities’ ESSI.

The term Enterprise Software System Infrastructure (ESSI) is used in this paper to denote enterprises’ portfolio of interconnected software systems. As the name implies, the term mainly addresses the software view of the systems, but also includes hardware related issues of the systems when necessary, since considerations like performance and availability in practice rarely are dealt with from a software perspective only. Further, systems are regarded as pieces of software, with externally accessible interfaces. All systems are conceptually divided into functional subsystems and databases, where the functional subsystems incorporate the functionality of the system and the databases serve as the data repositories. This division structures the analysis suitably in relation to the architectural problem areas identified in the electric utilities. The term component normally denotes a part of a system. In the context of the ESSI, however, the systems are the components. Therefore, the terms component and system will be used interchangeably. In this paper, the term architectural notation refers to diagrams or other constructs used to visualize architecturally relevant aspects of a system or an ESSI, such as structural and behavioral aspects.

Identified problems in the electric utility’s ESSI and ESSI management process are discussed in Section 2. In Section 3, the concept of software architecture, and notations used for architectural descriptions, are briefly presented. Section 4 argues for the need of a structured ESSI management process, and outlines some basic features of such process. In Section 5 established notations for architectural design of software product systems are evaluated in order to visualize the common problems of the ESSI presented in Section 2. The paper ends with conclusions in Section 6.

2. Problems with Electric Utilities’ Present Enterprise Software System Infrastructure

Swedish utilities have been operating on a stable market for a long time. This is partly because of well-defined market rules, and partly because of a stable technical process. However, the deregulation of the Swedish electricity market on January 1, 1996, has led to new requirements on the utilities, such as the division of the energy trading and network owning utilities into two legally separated organizations, one Distribution Network Operator and one Energy Producing and/or Trading Company. The operation and supervision of the transmission network and the overall balance in the network is the responsibility of the Swedish National Grid authority in cooperation with its counterparts in Norway, Finland, and Denmark. The regulating authority, NUTEK, carefully controls the Swedish Distribution Network Operators that remain monopolies within their areas of concession.

From a systems software engineering perspective, the deregulation process has created new demands on the planning, acquisition, and maintenance of utilities’ ESSIs. Requirements put up by NUTEK – including issues such as network balance settlement, reporting of network availability, and information separation in the divided utilities – has led to a need of adapting the utilities’ ESSIs. Furthermore, the utilities are going through a surge of mergers and acquisitions as a result of the deregulation, leading to a need of harmonization of ESSIs. Moreover, the Energy Trading Companies, now subject to the free market forces, find themselves in an acute need of software systems support for their entire business process.

In addition to the new requirements on the utilities’ ESSIs put up by the deregulation process, a number of technology-related factors contribute to the ESSI complexity. Electric utilities have a heterogeneous legacy of systems as to hardware platforms, operating systems and middleware software, compared to, e.g., the Manufacturing or Pulp-and-Paper Industry. According to the case studies carried out by the KTH research team, a typical system infrastructure of a Scandinavian mid-sized electric utility consists of as much as 100-120 interconnected systems, including administrative systems, technical systems, real-time control systems, management systems, and commonly a distributed office environment [1]. Present achievements in the fields of available middleware technology, in many cases triggered by the development of Internet related techniques and tools, open up rich opportunities to integrate new, and to extend the use of present, components of the utilities’ ESSI.

Several of the studied utilities have used “system maps,” i.e. informal line-and-box diagrams to visualize dependencies and interconnections between different systems and databases. Since these efforts have resulted in incomplete and/or ambiguous ESSI descriptions, the efforts spent have not supported the ESSI management process significantly. This paper focuses on notations for the description of an ESSI architecture. Therefore, the likewise important question of how a feasible architectural process in the enterprise can be accomplished, that closes the loop between the needs of the organization and the risks, opportunities, and restraints of software technology, is only briefly discussed.

The complexity caused by the legacy, present, and emerging technology, as well as the requirements raised by the deregulation process has an impact on utilities’ ESSIs. This section addresses three areas of architectural

---

2 Swedish National Board for Industrial and Technical Development.

3 Approximately 40 000 customers and 150 employees.
problems that reflect the impact of growing ESSI complexity, namely Overlapping and Shared Data, Overlapping and Shared Functionality, and Interface and Connectors.

Overlapping and Shared Data. Management of a utility’s data structure is a highly critical issue. Costs associated with the conversion of data between systems are often many times the cost of the systems themselves. Also, the most frequently used integration mechanism used by electric utilities is the sharing of data between different systems. Many systems in the ESSI are delivered with their own database. This means that the enterprise has limited possibilities of changing the physical structure of a database in the ESSI without loosing the prospect of upgrading the system to subsequent releases. Yet, the utility has a strong incentive to ensure consistency and integrity of the data structure. This can be accomplished, firstly, by requiring that the same information is only entered once in the ESSI, and secondly, by ensuring that accessed data is up-to-date in all ESSI databases.

Integration between heterogeneous relational databases is commonly implemented by the sharing or the exchange of flat ASCII text files, often by the use of low-level communication mechanisms such as FTP[5] and file formats such as EDI[6]. Also, database connectivity, widely used internally in systems, is used to provide real-time characteristics, commonly by the use of a 4th generation programming language that includes SQL[5] syntax. To gain complete control of the total data structure in all databases is not realistic, but the structure of the utility’s critical redundant data objects must be known to allow smooth database integration. Strategic considerations in this context include whether to take the cost for the implementation of a physical database that can be shared by several systems, or if several separate partially redundant databases are to be “glued” together. The first alternative gives a cleaner database structure, but might require tailor-made internal changes in several systems, whereas the latter alternative necessitates provision of functionality to update and convert data between the multiple databases.

Overlapping and Shared Functionality. An ESSI management issue that is becoming increasingly evident is the need for coordination of overlapping and shared functionality. One consequence of the development towards COTS[7] systems is that several systems in the ESSI include the same or similar functions. Overlapping functionality occurs when two or more systems can perform similar functions. One potential consequence is that seemingly similar functions may in fact differ significantly. For instance in the conducted case studies, details in functions for settlement on the deregulated market have been found to differ in how the algorithms are constructed. Furthermore, two overlapping functions do not necessarily update the same physical data, since the two functions might be allocated on different systems that operate on separate databases. These two problems address the need for visualization of the consistency, integrity, and synchronization of data in the affected databases, in order to provide information on whether to ensure the functional equivalence of the functions, or only permit the use of one of them. Another factor that adds to the complexity is the use of component-based systems that to a large extent utilize shared resources offered by operating systems, middleware platforms, and software development environments.

Interfaces and Connectors. A prerequisite for applying an external perspective on ESSI architecture, is the management of its interfaces. When a utility migrates its ESSI, modularity is a key issue. Ideally, systems in the ESSI should be considered as black boxes with well-defined interfaces. In practice, however, the interfaces in an ESSI are often poorly understood. The utility does not need to keep track of the details of all its interfaces, but in order to be able to replace components efficiently, the utility must be aware of the ESSIs interfaces – those used as well as those not used – and ensure availability of detailed information when needed. Since the ESSI components and their interfaces are heterogeneous, also the techniques used for interconnecting them, i.e. the architectural connectors [11], become heterogeneous. Middleware of many different types are commonly used. Therefore, in addition to the understanding of the interfaces of the ESSI components, an in-depth knowledge of the employed and the available connectors is needed.

The chosen granularity of the components in a utility’s ESSI is very much a strategic decision and should be based on the desired level of freedom of action. For instance, not only the external but also the internal interfaces of a system must be known if a part of the system is to be substituted later on without replacing the entire system in one single step. Information about internal interfaces is also required when implementing complex gateways [4] to bypass functionality in non-decomposable systems, or when ensuring consistency and integrity in databases that contain redundant information.

3. The Concept of Software Architecture

According to [11], software architecture involves the conceptual description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns. A sound basis for software architecture promises benefits
for both development and maintenance. Beyond the development stage, documenting a system’s structure and properties in a structured way has obvious advantages for maintenance. Understanding the existing system and the designer’s original intentions should help maintainers to preserve the system’s design integrity. For enterprises such as electric utilities, the state-of-the-practice of software architecture for ESSIs is pervasive in the form of informal diagrams and idioms. Unfortunately, these diagrams and descriptions are generally incomplete as well as highly ambiguous. Moreover, ad-hoc diagram types tend to focus on the components and not on the, from an architectural perspective, important interfaces and connectors.

According to [3] and [8], a system’s architecture may be modeled using standard design notations. In this paper, notations from Structured Methods (SM), Information Engineering Methods (IEM), and Object-Oriented Methods (OOM) are used to model the ESSI architecture. Structured Methods appeared in the late 1960s, incorporating a function-oriented view on system development. Next came the Information Engineering Methods with a focus on data rather than functionality. These methods were subsequently challenged by the emerging object orientation, now standardized in the form of the Unified Modeling Language. In this section, the basic features of a number of fundamental diagram types will be briefly presented. Notations may be classified into structural and behavioral modeling notations, where the structural notations model the static features, and the behavioral notations model the dynamics of the system. In this paper, we will focus on the structural modeling notations.

Entity-Relationship Diagram (ERD) [7]. The data model, presented in an ERD, consists of three interrelated pieces of information: the entities, its attributes, and the relationships that connect entities to one another. An entity, is an abstraction of a thing, an event, a role, a place, or a structure. An entity may be endowed with attributes, describing relevant properties of the object. Relationships are named connections between entities. Relationships are bi-directional, meaning that they may be followed in either direction. The cardinality specifies the number of objects on each end of the relationship, while the modality describes whether the relationship is optional or mandatory.

Class Diagrams [3] resemble entity-relationship diagrams in their basic structure. Instead of entities, the basic components are classes, i.e. descriptions of sets of objects with the same properties. Contrary to entities in ERDs, classes are not passive information containers; they may also perform operations. To reflect this, a class contains not only attributes, but also operations and semantics. Operations may be considered the equivalent to programming functions. The semantics specifies what the class does and how the class does it. To model the semantics, several approaches are possible, e.g. state machines or formal notation. Classes may also participate in relationships. In object-oriented methods, the most common relationships are dependency, generalization and association relationships. The dependency relationship denotes that one class uses another. The generalization relationship is a “is-a-kind-of” relationship. Association relationships between two classes denote a relationship where the classes are linked to each other in the sense that they may reference each other. Groups of classes may be organized in subsystems (or packages), to visualize their belongings.

Entity Diagrams. Orr’s Structured Requirements Definition (SRD) [6] makes use of Entity Diagrams that use circles, or bubbles, to denote entities, and arrows to denote data transferred between the entities. Entity diagrams are similar to, and sometimes also called, Data Flow Diagrams (DFD), but should not be confused with Yourdon’s DFD notation [10], in which bubbles denote transformations.

Jackson System Development (JSD) [6]. To describe the problem domain, JSD uses the concepts of entities and actions. An entity is more restrictive than that of ERDs or Entity Diagrams. The problem domain is presented in a tree-structure consisting of the entities and their associated actions.

Deployment Diagrams are used to model the topology of the hardware on which the system is executed. A node typically represents a processor or a device on which software components may be deployed. Like classes, nodes may participate in dependency, generalization, and association relationships. The relationship between a node and the component that it deploys can be shown explicitly by using a dependency relationship. Association relationships between nodes may represent physical connections, such as an Ethernet connection, a satellite link or a data bus.

Other Diagrams that are mentioned but not presented in this paper are State Diagrams [3], Sequence Diagrams [3], Data Flow Diagrams [10], and Context Diagrams [6].

4. Designing the Electric Utility’s Software System Infrastructure

An enterprise’s tool for dealing with the internal perspective, i.e. the content of an ESSI component, is a requirements engineering process, including the formulation of requirements, the acquisition of a system based on the requirements, and thereafter the validation of the new systems against the requirements. In order to give the software system vendor as much freedom as possible and thereby facilitate the use of its standard components,
the requirements are normally formulated as technology independently as possible. On the other hand, when dealing with the external perspective of the components in an ESSI, the enterprise seeks to organize its systems, i.e. ESSI components, to enhance the overall interaction, usability, and maintainability of its ESSI. It must therefore be stressed that dealing with the external perspective of an ESSI involves a large part of architectural design in contrast to the formulation of functional requirements. Thus, as the process of managing utilities’ ESSI becomes more complex, the structure that architectural notations may provide grows relevant also for organizations that use software systems.

There are a number of similarities and differences between the architectural design of software products and ESSI architectures. Enterprises do not have the problem of separating detailed design from architectural design [2], since no code normally is written. Instead, user organizations must carefully avoid specifying an architecture that implicitly or explicitly generates detailed technical requirements that could affect the freedom of action in single system procurements negatively. Hence, both software product architectures and ESSI architectures aim to describe an overall vision without unnecessarily delimiting the freedom of action for the design of its components. Notable differences are that the components in ESSI architectures tend to be larger and that both components and interfaces tend to be more heterogeneous. One reason for this is that the ESSI is generally the product of a large number of system acquisitions, replacements, updates, configurations, and development projects. Each modification to the infrastructure may also entail a modification to the architecture. Moreover, as in all design, the possible design choices are partially locked at the outset of each change of the ESSI. Some parameters are locked by the state of the art regarding software products and the availability of software products, and other parameters depend on the utility’s previous strategic design choices. The latter have been a special area of interest during the case studies since these have been found to be an underestimated issue from the utility point of view, i.e., seemingly insignificant technical decisions can later prove to be of significant importance for the overall ESSI.

To put the architectural notations in an ESSI architecture management context, there is a need to define some process-related concepts. Firstly, not only is there a need for a “blueprint” of the present ESSI architecture, but also for a description of the target ESSI architecture. The target infrastructure architecture is intended to describe the desired infrastructure, to establish a goal for the infrastructure management process. As the target infrastructure architecture is a long-term goal for the ESSI, the coupling to the enterprise IT strategy [5] is evident. Secondly, for the architectural description to be of use, it should preferably be used when the infrastructure is modified. The target infrastructure architecture description should ideally indicate the architectural delimitation and direction, while the present infrastructure architecture should highlight fundamental architectural predicaments.

A vital part of an ESSI architecture process is the identification of its strategic, or critical, ESSI architecture components. Identified components do not necessarily have to correspond with components in the enterprise’s present ESSI, but rather reflect the organization’s desired long term target architecture. Therefore, this paper employs the concept of Critical Data Objects and Critical Function Objects. Critical Data and Function Objects denote the data objects that are relevant both from a long-term architectural and from a business analysis perspective, such as customer, electricity circuit breaker, or contract. The Critical Data Object customer would, for instance, include attributes such as customer ID, name, address, etc. Critical Function Objects are the functional analog to Critical Data Objects, and can also be elicited in a business analysis process. Examples of Critical Function Objects are e.g. add customer and read remote meter. The purpose of the concept of Critical Data and Function Objects is to raise the level of abstraction when considering the ESSI target architecture.

To conclude, the total architectural description of an ESSI should provide the architect with an overall view of the present and future target architecture that can be used as a basis for strategic and tactical decisions regarding the utility’s ESSI. Examples of such decisions are the establishment of boundaries between ESSI components and the identification of possible simplifications in the structure of the ESSI.

5. Architectural Views of Electric Utilities’ Enterprise Software System Infrastructure

In this section, the architectural notations presented in Section 3 are evaluated against the utilities’ ESSI related problems discussed in Section 2. In analogy with [9], the notations have been visualized in five views related to the identified problems: The Data View describes the structure and the format of the data in the ESSI, The Functional View describes the functionality of the different systems in the infrastructure, The Interface View describes the relations between systems and interfaces, The Synchronization View describes the behavioral aspects of the interfaces, and The Deployment view describes the software disposition on the physical hardware topology. On account of its widespread use, its recent standardization, and its broad scope, the UML has been chosen as the representative notation for the object-
oriented paradigm. Notations from SM and IEM notations have been used as a comparison to the UML. It should be noted that the main purpose of the comparative analysis is to show that established notations for architectural design of systems also can be applied to the modeling of the deregulated electric utilities’ ESSIs. Therefore, the selected diagram types and the chosen views are to be seen as examples of how established notations for architectural design of systems can be applied on the modeling of ESSIs.

**The Data View.** For the utility, problems concerning overlapping and shared data may be visualized in class diagrams, entity-relationship diagrams, or SRD Entity Diagrams. The fundamental concern is to identify the data elements that exist in more than one database, what database updates another, and where the points of entry are located.

![Figure 1. Example of the Data View visualized with an UML Class diagram using Packages.](image)

Using class diagrams for the Data View, the classes are used to represent Critical Data Objects; the attributes represent the relevant database fields, while subsystems (or packages) represent the databases. Dependency Relationships represent the synchronization mechanism between data objects, where one Critical Data Object updates identical objects in related databases. Actors may be introduced in the diagrams, indicating the points of data entry. The dependencies between Critical Data Objects (classes) can be linked to the interfaces presented in the Interface View below, to specify through which interface the update procedure is realized.

Orr’s SRD Entity Diagrams may be used to specify the data update configuration. By using bubbles to represent the different databases, the arrows may be employed to specify the update structure. The main drawback of this notation is that the data structure of the databases is hidden in the diagram, and consequently does not support the identification of overlapping data.

![Figure 2. Example of the use of an SRD Entity Diagram for visualizing the Data View.](image)

**Entity-Relationship (ER) Diagram** may be used to model dependencies between databases. To visualize the Data View, the entities represent databases, while the attributes represent the Critical Data Objects of the database, and the relationships represent the data synchronization mechanisms. However, the basic entity-relationship diagram lacks support for two critical concerns. First, the attributes of the entity, representing data fields of the database, cannot be linked between entities. The exact data record that is updated in one system by another is thereby unspecified. Second, the relations in entity-relationship diagrams are bi-directional, implying that the direction of the update remains unspecified. ER diagrams may alternatively be used in analogy with class diagrams, where an entity represents a Critical Data Object instead of a database. The basic drawbacks, however, remain. A combination of SRD Entity Diagrams and ER diagrams may be used to substitute the class diagram option, as they cover the update structure and the content of the database respectively.

A data collection often overlooked is the configuration data of the systems constituting the ESSI. Neglect of the configuration data may prove expensive, as this information may be vital to the proper functionality of the ESSI, and may have required considerable resources to define. Adding this data to the ESSI data structure diagram enables a structured configuration data management.

**The Functional View.** Overlapping and shared functionality may present a problem much similar to the problem of overlapping and shared data. When similar functions are offered by different systems, it may be desirable to specify the function that is to be used, or to ensure the equivalence of all similar functions. Furthermore, for the configuration management of software releases, shared functionality, i.e. when for
instance two systems use a common software component, may be desirable to visualize. The main ESSI management issue is thus to identify overlapping and shared functionality when present. Class diagrams much like the ones used in the Data View may be employed to present the Critical Function Objects and their location in functional subsystems. Classes are then used to represent Critical Function Objects. Operations may be used to detail the Critical Function Objects, and subsystems may serve as functional subsystems. As in the Data View, actors may be introduced to indicate which functions that are used, and by whom. Association relationships can be used to connect similar functionality present in different functional subsystems, while dependency relationships may be employed to denote shared functionality.

![Graphical Information Functionality](image1)

**Figure 3** Example of the Functional View visualized with a UML Class diagram using Packages.

To link the Functional View to the Data View, functions may be linked to data objects that they operate on. Using class diagrams, the Functional and the Data Views may actually be presented in one single diagram, but to enhance readability and focus, the two views are separated. As with the Data View, ER diagrams may be used to visualize system functionality. Entities then represent the equivalents of the class diagrams subsystems. Attributes represent the Critical Function Objects, and the bi-directional relationships indicate similar functionality. Also in resemblance with the Data View, a drawback of this notation is that the relationships do not link attributes to attributes, as would be desirable in this context. Jackson System Development (JSD), DFDs and Structured Analysis and Design Technique’s Context Diagrams [6] may be used – not to describe the logical location of functionality – but to find the Critical Data Objects, their contents and their interdependencies. The identification of Critical Data Objects is however beyond the scope of this paper.

**The Interface View.** Interfaces are relevant for the ESSI architecture for three reasons. First, the possibilities of smooth integration of new systems depend on existing interfaces in both the present infrastructure and the prospective system. Second, opportunities of integration between already existing systems are made explicit. Third, external system integrators, suppliers and developers need this information in order to integrate new systems into the existing infrastructure. To visualize the logical interfaces between the systems, once again, class diagrams may be used. The systems – represented by the UML construct subsystems – with interfaces, specifies the operations supported by the respective systems. Systems using other systems’ interfaces are related to the interface by a dependency.

![Graphical Information System](image2)

**Figure 4.** Example of the Interface View visualized with a UML Class diagram using Packages.

A drawback of the UML notation is the implicit assumption of a common connector. Connectors specify the underlying services used for communication between systems and may for instance be COM®, DDE®, FTP, or some other type of middleware technology. In the heterogeneous ESSI, connectors are normally unique, which preferably should be made explicit in the notation. As mentioned above, the dependency relationships denoting data updates in the Data View, may be linked to the interfaces to make explicit through which interface the data passes. Time dependencies and message passing through interfaces can be visualized in the Synchronization View. The authors have not found any non-object-oriented notation to represent interfaces and

---

5 COM: Component Object Model.
6 DDE: Dynamic Data Exchange.
connectors adequately. Since interfaces between classes, components, and subsystems constitute an integral part of most object-orientated development, object-oriented notations have constructs for interfaces but lack easy-to-use notations for connectors, possibly because object-oriented development techniques commonly are intended for fairly heterogeneous systems in which the function and type of the connector is implicit.

**The Deployment View.** The Deployment view describes the hardware topology of the infrastructure, and visualizes the distribution of software components on hardware nodes. Ad-hoc line-and-box variants of this diagram type have been found in use at several of the investigated utilities. This is therefore stressed as the arguably most intuitive view, presenting the software distribution in its physical environment. Furthermore, it serves as the overall description of the ESSI, introducing links to the remaining views. Since the notation is straightforward, it is not credited to either OOM, SM, or IEM, even though it constitutes a diagram type in the UML.

**The Synchronization View.** Sequence diagrams and state charts can be useful for modeling behavioral aspects of the interface and connector mechanisms. Specifications with formal or semi-formal notations, for instance describing pre and postconditions, may also be used to specify the dynamics of the interfaces. However, this information easily becomes fairly detailed, and is therefore argued to be of less importance for an overall understanding of an ESSI architecture.

To conclude, object-oriented description languages such as UML have some marked advantages over the structured methods discussed in the paper. In particular, the inter-view linkages are more prominent using class diagrams, due to the adaptive constructs of the UML. This is an advantage compared to the evaluated IEM and SM diagram types that do not support couplings between the different views presented here. Since an architectural description language for ESSI architectures must be easy-to-use, only a relatively simplistic subset of the great variety of alternatives to express the structure and behavior of a system have been considered for the modeling of ESSI architectures. For instance, behavioral modeling has been considered to be of limited relevance in the context of the deregulated utility’s main problem areas. From an ESSI architectural perspective, much of behavioral modeling on this level of abstraction is the domain of business analysis or requirements engineering rather than infrastructure architecture. Even though the UML has decided benefits in this context, the application of object-oriented notations encounters problems when the differences between the ESSI and a single system become too prominent, e.g. in the case of connectors.

### 6. Conclusions and Further Work

This paper has evaluated how established architectural notations generally used for system development may be employed to support the Enterprise Software System Infrastructure (ESSI) management process of the electric utility. The authors suggest that applying established architectural notations to the ESSI can be beneficial as a tool for managing important architectural aspects. However, for an efficient and intuitive ESSI management process support, the development of specialized ESSI architecture notations constitutes further work. For example, none of the evaluated notations fully support the architectural concept of component connectors, which proves vital for modeling the heterogeneous nature of the electric utility’s ESSI. Aspects of overlapping and shared data and functionality may, however, be visualized successfully using object-oriented notations, such as UML class diagrams.

The presented ESSI architectural notations are of little use without a process to support the construction and maintenance of the architectural description. Important products of the management process, such as the target ESSI architecture and the Critical Data and Function Objects, which relate the ESSI architecture to the utility’s IT strategy and ultimately to its business goals, have only been outlined in the paper. A more exhaustive discussion on the management aspects of the ESSI and its architecture remain as further work.

### 7. References


