Modeling of Information Systems as Systems of Systems through DSM

Paolo Salvaneschi
University of Bergamo, Dep. of Management, Information and Production Engineering and
Salvaneschi & Partners
Bergamo, Italy
paolo.salvaneschi@unibg.it

ABSTRACT

Information systems may be composed of dozens of software applications managed by different teams knowing only part of the whole system. During the evolution, the structure of information systems tends to become more and more complex. This opens the problem of understanding and managing the *Information System-level* architecture where a software component is a whole software application.

We present the preliminary results of a project aiming at modeling information systems as Systems of Systems via Design Structure Matrices (DSM).

The information system is modeled as a matrix describing components (software applications) and connectors (relations between applications). The model, implemented with the Cambridge Advanced Modeler tool, allows us loading the matrix and visualize it as a graph of nodes and oriented arcs. Different classes of users may be interested in different views of the model. A view originates specific attributes of components and connectors. The tool allows us querying the model highlighting nodes and arcs of interest for a specific view.

The case studies – composed of about a hundred software applications – consist of the information systems of a retail company and of an hospital.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

Keywords

Information System, System of Systems

1. INTRODUCTION

Large information systems are composed of dozens of *software applications* – programs that typically implement a business process or part of it. Applications may be developed in house or acquired from vendors and possibly adapted. During the evolution the information system grows, integrating more and more applications and changing the existing ones. The evolution is managed by different vendors and development teams working only on parts of the whole system.

Due to time pressure and lack of global knowledge, local changes may cause decay of modularity and increasing complexity of relationships between software components. Maintainers experience a growing difficulty to understand the whole system.

This opens the problem of understanding and managing knowledge about the *Information System-level* architecture where a software component is an entire software application.

Yet diverse groups of professionals are interested in maintaining different aspects of such knowledge of the information system. Business people want to know the relationship between business processes and software applications. Developers and system integrators manage software changes and need to know the relations between applications. The architecture office wants to monitor the complexity, discover architectural smells and verify architectural rules.

In this paper we present the preliminary results of a project aiming at modeling information systems at the abstraction level of Systems of Systems. Our case studies consist of the information systems of a large retail company and of an hospital. Both systems are composed of about a hundred software applications.

The information system is modeled as a Design Structure Matrix [1] of components and connectors (software applications and relations between them). The model, implemented through the Cambridge Advanced Modeler tool [8], is incrementally defined through the identification of classes of users. Each class requires different views of the model and each view leads to the definition of specific attributes of components and connectors.

The tool allows us to visualize the model as a graph of nodes and oriented arcs. Both nodes and arcs may be qualified by attributes. Also, the model supports different views. For example, components connected to a selected component and types of connectors; components and connectors implementing a business process; components and connectors related to an architectural smell.

In summary, the paper makes the following contributions:

- We investigate modeling an information system at the abstraction level of a System of Systems.
- Based on this technique, we present models of real world information systems using a Design Structure Matrix and a tool to visualize views of the matrix as graphs of components, connectors and attributes.
- We discuss positive aspects of this approach as well as open problems and suggest future research directions.

The paper is structured as follows. Section 2 presents the information systems that are the object of this study. Section 3 defines the modeling technique. Section 4 describes how the model is implemented using the Cambridge Advanced Modeler tool. Section 5

SESoS'16, May 14-22, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897829.2897832

defines the structure of the patterns used for exploiting the model and presents some examples. Section 6 discusses the status of the project and future directions. Section 7 provides an overview of related work.

2. CASE STUDY CONTEXT

The case study involves two different information systems.

The first study (*System 1*) is related to the information system of the Italian branch (more than 8000 employees) of a large European retail company. IT department (70 employees) manages about 100 software applications and three large databases. About two thirds of applications are developed according to the company specification (about 500.000 LOC of Java-JSP code and about 1.100.000 LOC of RPG code). The remaining third is composed by software products acquired from vendors and adapted/integrated. The main development effort is outsourced to external suppliers.

The second study (*System 2*) is the information system of a hospital in North Italy (4000 employees). IT department (30 employees) manages a system of about 100 software applications. Almost all applications are software products acquired from vendors and adapted/integrated.

3. MODEL

In this section, we describe the conceptual structure of the model that is implemented through a Design Structure Matrix. The aim is to model the information system at the abstraction level of a System of Systems. A component of the model is a software application.

Conceptual structure.

The conceptual structure of the model is composed of:

- *Classes of users*. Each class represents a group of stakeholders with specific interests in exploiting the knowledge about the information system modeled as a System of Systems. An example is the class of software developers/integrators.
- *Views*. A view is a specific subset of model elements and associated attributes. A view is a representation of one or more aspects of an architecture [3] that illustrates how the architecture addresses one or more concerns held by a group of stakeholders. For example, developers/integrators want to view the data flow connections between applications.
- *Components and connectors*. The architecture of the information system at the abstraction level of a System of Systems is represented by two classes of objects: components and relations between components. The elements of the two classes may be enriched by attributes according to the views. For example, the connections between applications may be qualified by the *type of connector* (FTP, JDBC, RMI,...).

Classes of users and views.

Different classes of users may exploit the information system model for different purposes. We identified the following classes:

- *Process oriented users*. They are interested in knowing the interaction between business processes and software applications.
- *Developers and providers*. They need to know the relations between applications to be changed or integrated.
- *Architects*. They are interested in analyzing the information system to discover symptoms of architectural degradation and violations of architectural rules.

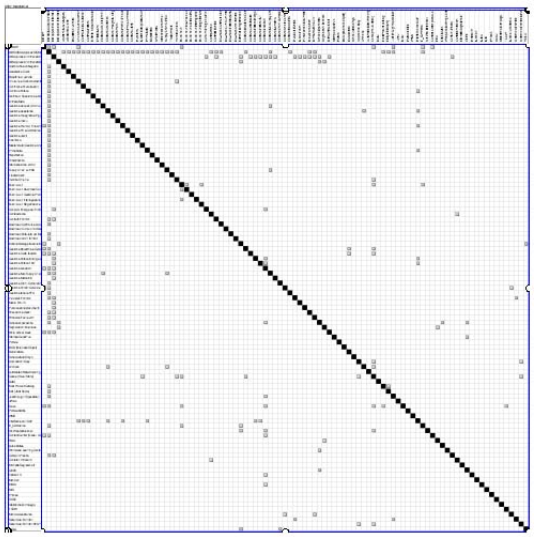

Figure 1: DSM - System 1.

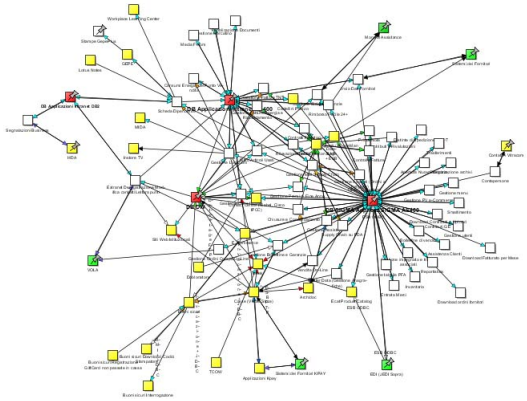

Figure 2: Graph visualization - System 1.

Each class of users may require different views of the information system model. We identified the following views:

- Software applications and data flow relations between applications.
- Software applications of a specific type *shared database* and data flow relations with other applications.
- Types of connectors between applications.
- Architectural smells.
- Mapping between business processes and software applications.

Each view may be useful for one or more users classes.

Components and Connectors.

The core of the model is a Design Structure Matrix of Components and Connections. For example, Figure 1 is the DSM of *System 1*. Rows and columns list all the software applications. A shared database is modeled as a software application. A grey cell crossing the column of application A and the row of application B in the matrix is an oriented relation between A and B. The default meaning of the relation is: there is a flow of data from A to B.

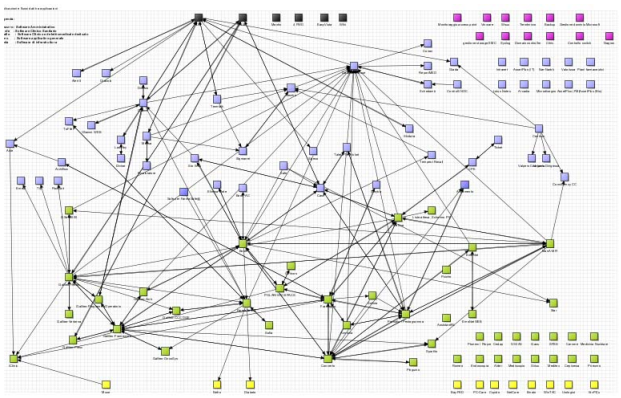

Figure 3: Graph visualization - System 2.

Figure 2 is the visualization of the DSM of *System 1* as a graph. A node of the graph is a software application and an oriented arc is a connection between two applications. Both components and connectors may have attributes. For example, node A is connected to node B. Both A and B hold the name of a software application. The oriented arc between A and B may have an attribute defining the connector type (e.g., FTP or enterprise bus or RMI invocation).

Again, nodes A and B and the arc between them may implement a business process. In this case, the name of the process is an attribute of A and B and the data flows names may be arc attributes (e.g., orders or invoices).

Attribute values may be visualized through different colors like in Figure 2. Both node and arc attributes are defined according to the views required by the classes of users.

Figure 3 is the visualization of the DSM of *System 2*. Both *System 1* and *System 2* graphs show a quite complex System of Systems architecture. Even a bird's-eye view is sufficient to highlight significant differences. *System 1* (the retail company) exhibits a number of hubs – the shared databases. Other applications center around hubs. In *System 2* (the hospital) the topology is more uniform: it consists of a network of acquired and integrated applications and a set of stand-alone applications.

Model quality.

The quality of the model depends on the quality of information collected to develop it. In our project the information was derived manually from the documentation of software applications and knowledge of application owners.

This is a significant area of improvement. The support of tools to automatically extract information from code is certainly valuable, but hardly implementable in practice in an industrial context – for example, the information systems of the study include many software technologies as well as COTS executable components.

During the evolution of the information system, the model has to be changed and the quality must be guaranteed. The maintenance effort is not high because the size of information included in the DSM model is limited. The important issue is the implementation of an evolution process that guarantees the communication of information related to changes.

4. IMPLEMENTATION

The models are implemented with the Cambridge Advanced Modeler (CAM) tool. The tool allows us to load the DSM and to visualize a matrix as a graph. Both nodes and arcs may be decorated with attributes.

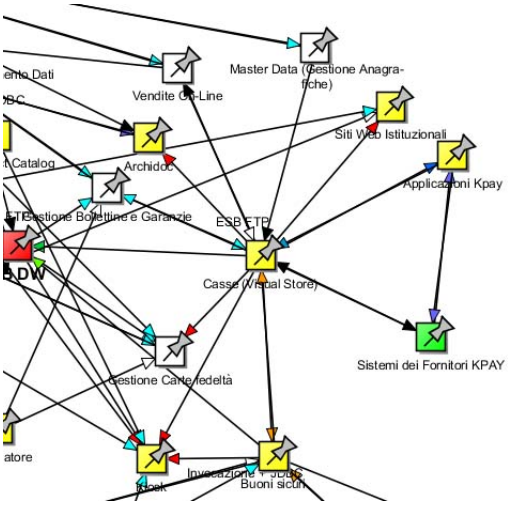

Figure 4: Smell - System 1.

It is also possible to define specific views of the graph through the selection of different attributes and the visualization through different colors. These predefined visualization criteria may be cataloged and recalled from a list of model queries.

5. QUERY PATTERNS

The model may be used by different classes of stakeholders through different views with specific attributes of the basic component-connector model and specific ways of visualizing the graph. We organized all these elements into *query patterns* – predefined ways to visualize the model according to specific concerns.

A pattern includes:

- Pattern name.
- Classes of users.
- View and model attributes.
- Model query.
- Explanations and suggestions.
- Example.

We defined the following patterns:

- Applications and data flow relations between applications.
- Databases and data flow relations with other applications.
- Applications and type of connectors between applications.
- Architectural smell: an application is connected to many other applications.
- Architectural smell: an application is connected through many types of connectors.
- Architectural smell: an application is connected to many databases.
- Architectural smell: a database of type data-warehouse has output connections with applications.
- Applications and connectors implementing the business process A, B,...

Figure 4 shows an example of an architectural smell in *System 1*. The application in the center of the figure is connected to many other applications. The arrows of the connections are color-coded according to type of connector. The figure shows that the application is connected through many types of connectors. The smell suggests

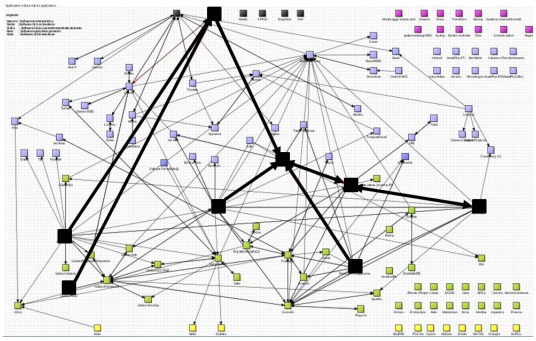

Figure 5: Business process - System 2.

a local complexity that has to be evaluated and possibly may be re-designed.

Figure 5 shows the visualization of applications and connectors implementing a business process. The highlighted elements of the information system of the hospital – *System 2* – are involved in the interaction with the regional health care system.

6. DISCUSSION AND OUTLOOK

In both the studies the construction of the DSM model required a significant effort.

In *System 2* the available documentation for the software applications was very poor. In this case, the knowledge was collected through the interviews of the application owners.

Luckily the documentation process for *System 1* is much more structured. A wiki maintains the key information to support software evolution. Each software application has an architectural document that includes the relations with the other applications.

In both cases the management supported the project and results received highly positive feedback. The requirement to model the information system at the abstraction level of a System of Systems is evaluated as important in both IT organizations.

In health care IT – *System 2* – models are considered an effective instrument to reason about integration of software from multiple vendors limiting costs and risks of the integration process.

In retail IT – *System 1* – models are viewed as a tool to manage the growing complexity of the system in the evolution process. IT managers have an increasing awareness of the lack of concepts, models and tools to support knowledge management and decision making at a higher abstraction level than a single software application. The information system evolves towards an increasing complexity. The number of applications – both custom and COTS-based – grows. Also, over time, more and more projects include different providers and development teams. Each provider and team has a local knowledge and the time/cost pressure forces concentrating the effort on local optimum performance. The lack of global knowledge increases the risks of failures and the costs due to re-working activities.

Future directions.

An open problem is the limitations of the class of tools we used: they are able to store, process and visualize DSMs, but don't offer the full functionality of a database management system and flexible navigation of the graph. Also, with the tool we adopted, it is not easy to associate complex attributes to the elements of the matrix and the ability to query the graph is limited.

To solve these problems we are experimenting the use of graph-databases. This approach allows us storing complex data structures as well as access them with a query language to retrieve the model – more easily extracting and visualizing graphs.

Another improvement is the development of new patterns including other classes of users like support service teams. The idea is to catalog for each pattern a set of predefined graph queries to support the most common uses of the model.

7. RELATED WORK

DSM have been applied in different software contexts. La Mantia [4] and others examine the modularity structure and the design evolution of two software product platforms using DSMs. In their analysis the DSM components are at the abstraction level of a class.

Huynh and others [2] present an automated approach to check the conformance of implemented modularity to designed modularity, using DSMs. The architecture conformance problem is also examined by Passos and others [6]. The illustrative application in the paper uses a DSM to show the adherence of the code to the MVC pattern. Tekinerdoğan [7] introduces the use of DSMs in the context of scenario-based architecture analysis methods.

Merkle [5] examines the architectural erosion of software applications and the reasons why erosion occurs. The paper provides an overview of architectural visualization and analysis tools, including DSMs. The tools may also be used to visualize violations of design rules (for example, layering rules).

In all these works DSMs have been typically used as a tool to extract architectural knowledge from code and support the architectural analysis of software in the context of a single software application. To the best of our knowledge, no previous research nor reported industrial experience is available related to the application of DSMs at the information system-level architecture.

8. REFERENCES

[1] S. D. Eppinger and T. R. Browning. *Design structure matrix methods and applications*. MIT press, 2012.

[2] S. Huynh, Y. Cai, Y. Song, and K. Sullivan. Automatic modularity conformance checking. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 411–420, New York, NY, USA, 2008. ACM.

[3] P. Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12(6):42–50, nov 1995.

[4] M. J. LaMantia, Y. Cai, A. D. MacCormack, and J. Rusnak. Analyzing the evolution of large-scale software systems using design structure matrices and design rule theory: Two exploratory cases. WICSA 2008, pages 83–92, 2008.

[5] B. Merkle. Stop the software architecture erosion. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '10, pages 295–297, New York, NY, USA, 2010. ACM.

[6] L. Passos, R. Terra, M. Valente, R. Diniz, and N. Mendonca. Software architecture conformity static architecture-conformance checking: An illustrative overview. *IEEE Softw.*, 18:82–89, 2010.

[7] B. Tekinerdogan, F. Scholten, C. Hofmann, and M. Aksit. Concern-oriented analysis and refactoring of software architectures using dependency structure matrices. Early Aspects Workshop AOSD, pages 13–18, 2009.

[8] D. Wynn, D. Wyatt, S. Nair, and P. Clarkson. An introduction to the cambridge advanced modeller. In *Proceedings of the 1st International Conference on Modelling and Management of Engineering Processes*, MMEP 2010, 2010.