In this article we show how knowledge representation techniques can be applied for software architecture. We define a representation model for software architecture concepts. The model is based on MY model (meta modeling in Y), which is a knowledge engineering methodology. It represents software architecture concepts using three branches: component, connector, and architecture. The component branch represents concepts that are related to computations, the connector branch represents concepts that are related to interactions, and the architecture branch represents concepts that are related to the structure and the topology of the described system. We think that such a representation of architecture concepts aids in improving reusability not only in the implementation level, but also in the description level. The model assigns a hierarchical library for the four software architecture conceptual levels (meta-meta architecture level, meta architecture level, architecture level, application level).

Keywords: Knowledge representation, software architecture, meta modeling, MY meta-model, components, connectors, architectures.

1. Introduction

Software architecture concerns with constructing software systems based on components (encapsulation of functionality) and connectors (encapsulation of communication) [1] [2] [3] [4] [5]. By doing this, it shifts our focus of developing software systems from line-of-code to a higher level, which is a coarser-grained architecture element. There are many benefits of this including improving reusability, enhancing understandability, and reducing costs. In software architecture, systems are described as a collection of components that interact with each other using connectors; the topology of a system is represented by a graph of components and connectors. Therefore, the three main players of software architecture are components, connectors and architectures (sometimes called configurations, systems, or topologies).

Modeling, meta modeling and reusable component libraries are proven techniques that are used in Knowledge Engineering (KE). Modeling and meta modeling techniques are used to represent systems at a high level that abstracts implementation considerations and
focuses instead on competence [6]. In the domain of Knowledge Representation one speaks about meta-knowledge to evoke knowledge relating to knowledge, or meta-model for a model representing a model. Meta modeling knowledge representation technique is widely used for integration and definition of models from different domains and for standardization purposes. It is a promising way for the definition of different models describing a system.

Classic KE methodologies such as CommonKADS [7], TINA [8] VITAL [9], MIKE [10], Protégé [11], Generic Tasks [12], Components of Expertise [13], and MY [14] describe Knowledge Based Systems (KBS) using three distinct and complementary components: task, Problem Solving Methods (PSMs), domain. Separating the methods (used to achieve a given task), the task, and the domain permit to define a KBS application as a combination of these three components. The MY meta modeling approach [14] [15] [16] combines meta modeling and reusable component libraries. A KBS in MY model is described according to three axes:

1. object of reuse axis, determines the minimal and the sufficient components that are needed to describe a KBS.
2. level of granularity axis, allows to describe the KBS components according to different levels of granularity for clarity and reuse purposes,
3. process of reuse axis allows describe components for reuse or by reuse. The MY model uses a horizontal and vertical decomposition on a given system in order to capture its complexity and to identify its components.

The model uses horizontal and vertical decomposition to capture the system’s complexity and to identify its components. Moreover, the model has a multi hierarchal library to improve the reusability of the different concepts.

In this article we show how knowledge representation techniques can be applied to software engineering. In particular, we explain how the MY meta modeling approach, which is implemented by our research group [14], can be used in software architecture. The model we present is not an architecture description language (ADL), but it is a model to guide and help the architect in the process of describing a software system.

The remainder of this article is organized as follows: Section 2 explains the model Y and its applicability to represent the concepts of software architecture. Section 3 presents the three axes of software architecture, conceptual level axes, reusable architecture elements axes, and process of reuse axe. Section 4 gives an example of representing a client-server systems using the model. Section 5 concludes this article.

2. The MY model

Using the MY model, the architecture of a system is described by three aspects: component, connector, and architecture.
1. component, to represent computations,
2. connector, to represent interactions,
3. architecture, to represent topologies and structures.

These three concepts are joined together to build a software architecture. They are defined with an aim to reuse them in describing different systems. Therefore, we can reuse a model to describe different systems of different domains.

- a component can be used to describe a server, a client, a shared data file, etc.
- a connector can be used to describe a function call, an RPC, pipe, etc.
- an architecture can be used to describe a client-server systems, a pipe-filter architecture, etc.

We think that a system can be well described using this model, which has the ability to represent all aspect of software architecture.

2.1. Main concepts

After examining different literatures on software architecture [5] [17] [18] [19] [20], we think that the three main concepts of architectural description are: components, connectors, and architectures. Moreover, these three concepts represent different architectural elements, for example, components can represent concepts that are related to computation: components, ports, roles, etc, connectors can represent concepts that are related to interactions and associations: connectors, bindings, attachments, etc, and architectures represent concepts that are related to the topology of systems: architectures, configurations, systems, etc. We expose these concepts using three aspects each represented by a branch of the letter Y, Component aspect, Connector aspect, Architecture aspect.

Hence, each branch of the Y describes a concept of the architecture (as shown in figure 1). For example, the component aspect makes it possible to describe generic components. The more the component is far from the center of the Y architecture, the more it is reusable. The center of the Y architecture describes a primitive component associated with a primitive connector (included in the component) and described in the terminology of an architecture.

In addition, in order to keep the description of the three concepts (component, connectors, architecture) independent from each other, we introduce two relations: inter-link and intra-link. The relation inter-link describes the link between two different concepts (e.g. attachment between a component and a connector), while the relation intra-link describes the link between two concepts from the same type (e.g. binding between two components, binding between a composed architecture and its sub-architectures). These relations are used in the model as adapting and assembling tools for describing the topology and the structure of systems.
2.2. Multi-Abstractions and Multi-Views

In order to capture all aspects of systems and to encapsulate their complexity the model MY describes models according to two decompositions: vertical decomposition or multi-abstractions and horizontal decomposition or multi-views (cf. figure 1).

2.2.1. Multi-abstractions

Multi-abstraction defines different abstractions for a given element. Sometimes, an architecture is described using different levels of abstraction. For example, refinement across different abstractions (abstract level, concrete level, and implementation level [21]) can be represented using the Y architecture with multi-abstractions; each refinement level is represented by a circle starting from the higher level to the lower level. Another example is the architectural description of systems using ADLs that define connectors implicitly (e.g. Darwin [22]). In these case the highest circle (representing the highest abstraction level) shows only a component and an architecture and we need to zoom in (to lower abstraction levels) in order to find the connector (as shown in figure 2). Moreover, to get a simplified model with less number of elements we zoom out (to a higher abstraction levels).
2.2.2. **Multi-views**

Multi-view allows the definition of different views for a given system. Having different views clarifies different important aspects and helps managing the complexity of software engineering artifacts (such as requirements specifications and design models), separation of concerns. A representative example of a work with multiple views in software architecture is provided by Hofmeister *et al* [23]. They present four views; each has particular elements that need to be described. The four views are the conceptual view, the module view, the execution view, and the code view. The conceptual view describes the architecture in terms of domain elements; here the architect designs the functional features of the system. The module view describes the decomposition of the software and its organization into layers. The execution view is the run-time view of the system: it is the mapping of modules to run-time images, defining the communication among them, and assigning them to physical resources. The code view captures how modules and interfaces in the module view are mapped to source files, and run-time images in the execution view are mapped to executable files. However, we think that the number and the representation of views depend on the system being described and the different stakeholders interested by this system.

The MY model represents different views using multiple Ys, each Y represents a view of the system and how the different elements (components, connectors, architectures) are supported in this view (explicitly or implicitly). For example in the system shown in figure 3 components, connectors and architectures are explicitly represented (described) in the conceptual view and the module view, but in the execution view and the code view only components are explicitly represented, while connectors and architectures are implicitly represented (it must be noted that all the views are seen at the same level of abstraction).
3. The three axes of software architecture

The modeling of software architectures has led us to define architectures according to three dimensions or axes: Conceptual Level axis, Reusable Architectural Elements axis, and Process of Reuse axis, as given in figure 4.

3.1. Conceptual level axis

The conceptual level axis represents the different modeling levels of the architecture elements, from the meta-meta-architecture level to the application level. Inspired by the four modeling levels of OMG [24], we define four conceptual levels for software architecture: meta-meta-architecture, meta-architecture, architecture, and application. The Y architecture will be applied to each level taking into account the architectural elements correspond to that level only.

We exhibit in this axis these levels (as explained by the schema in figure 5 using the example Client-Server given in [17]):

1. Meta-meta-Architecture level (M²A): provides the minimal elements of architectural modeling. It is represented by the three branches of the form Y: Meta-component, Meta-Connector, Meta-Architecture. These three elements are the base for defining different meta-architectures. A meta-meta-architecture conforms to itself (instance of itself). An example of a meta-meta-architecture model is MADL [25].

2. Meta-Architecture level (MA): provides the basic modeling elements for an architecture description language: Component, Connector, Architecture, Ports, Roles, etc. These elements are the base for defining different architectures. Meta-architectures conform to meta-meta-architectures. In the scope of a conformity relation, each element of MA is associated with an element of M²A. For example, in figure 4 component is associated with meta-component, connector is associated with meta-component and meta-connector, and configuration is associated with meta-component. However, these three architectural elements differ from one ADL to another, e.g. in Acme [17] and COSA [26] we find not only components, connectors, and configurations, but
also styles, ports, roles, bindings, attachments, properties, and constraints. While in SOFA, components are Frames and configurations are presented as components [27].

Fig. 4. The three axes for software architecture.

3 Architectural level (A): in this level, various types of components, connectors, and architectures are described. These various elements are, in general, stored in a library of reusable elements. Architectures conform to meta-architectures (ADLs), therefore each A element is associated with an MA element. For example, Client and Server are components, RPC is a connector, and Client-Server is a configuration.
Application level ($A_0$): allows us to describe applications. An application is seen as an assembly of instances of types of components, connectors, and architectures. Applications conform to architectures. Each element of $A_0$ is associated with an element of $A$. For example, CL1 is an instance of Client, S1 is an instance of Server, RPC1 is an instance of RPC, and C-S is an instance of Client-Server.

![Software architecture conceptual levels with UML notations, using the example Client-Server](image)

**Fig. 5.** Software architecture conceptual levels with UML notations, using the example Client-Server

### 3.1.1. Hierarchical Library for reusable elements

Modeling complex systems requires that the independent information related to each conceptual level be stocked in a library. Reusable component libraries have several benefits including communications, interoperability, browsing, searching, systems engineering and guidance [28]. Reusable component libraries have been described in different KE approaches. After examining these libraries, we can see that they are facing the same problems amongst: how to describe a component (descriptive language), what should be a component’s level of granularity, how to structure and index components so
that users can easily understand, find, configure and integrate components in order to construct their applications.

The library in the MY model is a multi hierarchical library with four different levels: meta-meta architecture library to stock meta-meta elements such as meta-components, meta-connectors, meta-interfaces etc, meta architecture library to stock meta elements such as components, connectors, ports, etc, architecture library to stock types of architecture elements such as server, client, pipe, filter, RPC, etc., and application library to stock instances of architecture elements. End user can use (and reuse) this library to describe the architecture of its system. The issue of reusability is an essential issue in software architecture, where elements (component, connector, etc.) should defined as a reusable concept for other applications.

3.2. Reusable Architectural Elements axis

This axis makes it possible to identify reusable architectural elements for each conceptual level. These reusable elements are then stored in a library associated with each conceptual level and used by its inferior level. For example, the reusable elements of level MA are components, connectors, configurations, ports, and roles. They are stored in the library and used by architectures of level A.

Four categories of users involve in the process of reuse, each user is responsible for a conceptual level. In addition, a fifth user is needed to manage the library. These users are:

1. CI (Constructor of the Infrastructure): defines the architectural basic concepts on which the Constructor of the Architectural Language (CA) bases to build an architecture description language (defines the modeling components that are used to build an infrastructure). The CI is concerned with the conceptual level M^A.
2. CA (Constructor of Architectural Languages): defines the corresponding ADL by instantiating the concepts provided by CI. The CA is concerned with the conceptual level MA.
3. AA (Architect of Applications): its role is to conceive and describe the architecture, which will be used for the development of the application (describes the architecture of the application). The AA is concerned with the conceptual level A.
4. DA (Developer of Applications): guides an architecture in the process of implementing the application. In certain cases, one can automatically generate the code starting from the architecture and in this case the DA does not practically intervene. The DA is concerned with the level A_0.
5. ER (Manager of the library or Engineer of the Reuse): builds and manages the whole library. The library is composed of generic meta-components, meta-connectors, and meta-configurations defined by CI, components, connectors,
configurations, ports, roles, etc defined by CA, types of components, connectors, and configurations defined by AA, and instances of components, connectors, and configurations defined by DA. The library has four different parts: meta-meta architecture library, meta architecture library, architecture library, and application library. ER is concerned with all conceptual levels.

Figure 6 presents a UML sequence diagram for a possible scenario of the different operations that can take place in the library. This scenario is composed of three steps:

1. Construction of an ADL: During this step CA sends a demand for elements of level M^2A to ER to define an ADL (phase 1). Next ER searches in the level M^2A of the library for the elements, gets them (phases 2,3) and then the elements are offered to CA (phase 4). At the end CA responds with an acknowledgment (phase 5). If the elements are not available in the library, ER demands CI to construct these elements.

2. Definition of an architecture: In this step AA sends a demand for elements of level MA to ER to define an architecture (phase 6). Next ER searches in the level MA of the library for the elements, gets them (phases 7,8) and after that the elements are offered to AA (phase 9). Then AA confirms with an acknowledgment (phase 10). If the elements are not available in the library, ER demands CA to construct theme.

3. Definition of an application: DA can take part in the operation in order to guide the architect in the process of describing an application. In this case, DA sends a demand for types of elements to describe an application (phase 11). Next ER searches in the level A of the library, gets these types (phases 12,13) and then the types are offered to DA (phase 14). Finally, DA responds with an acknowledgment (phase 15). If the required types are not available in the library, ER demands AA to construct them.
3.3. **Process of Reuse axis**

The process of reuse concerns with the description of architectural elements following a process of reuse. This process consists of two steps:

1. Engineering for reuse, makes it possible to specify elements that are built for reuse for each conceptual level,
2. Engineering by reuse, makes it possible to specify elements that can be built by reuse.

Thus, this third axis allows architectural elements to be seen according to their construction (for reuse) in order to publish the library, and their use by selecting them from the library (by reuse) to build new architectural elements or even new applications.

The process of reuse has two steps:

1. Engineering for reuse, where reusable elements are identified according to their conceptual level, then described using a formal, semi-formal, or descriptive language and finally integrated in an associated library by choosing a well defined method (hierarchized by type, size, contents, frequency of reuse, etc). Engineering for reuse has three phases:
• **Identify.** In this phase user needs are identified. For example, in the conceptual level M²A one can identifies the need for meta-components, meta-connectors.

• **Represent.** After architectural elements are identified, they must be represented. Formal, semi-formal, or descriptive languages are used.

• **Organize.** Architectural elements are then organized in the suitable library’s layer depends on their conceptual level. The M²A layer, MA layer, A0 layer, or A layer of the library.

2. Engineering by reuse, where search and select of reusable elements are made according to needs, which are expressed by different categories of users. The technique of searching depends strongly on the organization of the elements in the library as well as the language chosen during the preceding step. The selected elements are adapted, if need, and integrated in the architecture. Hence, engineering by reuse has three phases:

• **Search and select.** It concerns with searching in the library and selecting the needed elements.

• **Adapt.** It concerns with adapting an architectural element to one’s needs. Adapting an architectural element is a hard task, it can be achieved using mechanisms such: instantiation, composition, generalization/specialization, etc. These mechanisms represent intra-links among the same type of concepts.

• **Use.** Finally the architectural element is ready for use and can be integrated in the appropriate conceptual level.

4. **Demonstrative example**

In this section we treat the well-known client-server system using the Y model to illustrate how our model can be used in the domain of software architecture.

The example (shown in figure 7) describes the architecture of a client-server system with a server composed of the components (connection manager, security manager, database) and the connectors (SQL query, clearance request, security query) and a primitive client. The client requests services and the server provides services, they interact using a connector (RPC). This example is very well know in the software architecture community and has been treated by most of ADLs [17].
4.1. Representing the example client-server using MY (at architectural level)

To represent this example using the model MY, we could have a MY model for each conceptual level. For example, for the architectural level (level A) we have two levels of abstraction:

- a high level of abstraction with the components server and client, the connector RPC, and the architecture client-server,
- a low level of abstraction with the components (ConnectionManager, SecurityManager, Database) and the connectors (SQLQuery, ClearanceRequest, SecurityQuery), as shown in figure 8.

Moreover, the MY model for this example could have two views, as shown in figure 9:

- module view with a component, a connector, and an architecture,
- code view with only components, where most of implementation languages do not separate components from connectors.
4.2. The client-server example according to the process of reuse.

In order to clarify how the process of reuse works and how the different users of the library intervene, we try to construct a library for the example client-server starting from the meta-meta architectural level to the application level.

4.2.1. Engineering for reuse

In this phase elements are identified according to their conceptual level, next described and finally integrated in the library.

Meta-meta architectural level

1. Identify. The reusable elements of the meta-meta level are first identified. For the example of client-server we can identify four meta elements: meta-components to define components, meta-connectors to define connectors, meta-
interfaces to define ports and roles, and meta-architectures to define configurations.

2. Represent. After meta-architectural elements are identified, they must be represented. We can use an object-oriented language to represent the meta elements as meta classes.

3. Organize. Finally, the meta-architectural elements are organized in the M2A layer (meta-meta-architectural layer) of the library.

Meta architectural level

1. Identify. The reusable elements of the meta level are first identified. For the example of client-server we can identify the following architectural elements: components to define architectural elements related to computation, connectors to define architectural elements related to interactions and associations, ports and roles as interaction points, and configurations to define topologies.

2. Represent. After the elements are identified, they must be represented. Architectural elements can be represented as generalizable classes. For example we can see the representation of components as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>component’s name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>provide, require, both</td>
</tr>
<tr>
<td>Non-functional properties</td>
<td>yes/no</td>
</tr>
<tr>
<td>Constraints</td>
<td>yes/no</td>
</tr>
</tbody>
</table>

3. Organize. Next, the architectural elements are organized in the MA layer (meta architectural layer) of the library.

Architectural level

1. Identify. Types of architectural elements are first identified for reuse. For the example of client-server we can identify the following types of architectural elements: server as a type component, client as a type component, RPC as a type connector, client-server as a type configuration.

2. Represent. After the types of architectural elements are identified, they must be represented. Types of architectural elements can be represented as specialized classes.

3. Organize. The types are then organized in the A layer (architectural layer) of the library.

Application level

1. Identify. Types of architectural elements are first identified and then instantiated. For the example of client-server we can identify the following instances: S1 as an instance of the component type server, C1 as an instance of the component
type client, RPC1 as an instance of the connector type RPC, and arch-1 as an instance of the configuration type client-server.

2. Represent. Application elements are represented as instances of architectural elements.

3. Organize. The application elements are then organized in the $A_0$ layer (application layer) of the library.

4.2.2. Engineering by reuse

In this phase elements are selected and reused by different categories of users.

Meta-meta architectural level

1. Search and select. If, for instance, the CI (Constructor of Infrastructure) wants to construct new ADL, it searches for meta elements in the library and selects them.

2. Adapt. The CI can use these meta elements after adapting them regarding the requirements and regulations of the new ADL. For example, it may define a new meta-component by composition.

3. Use. This meta-component then can be used by the CA (Constructor of Architectural languages) to define new architectures.

Meta architectural level

1. Search and select. The CA (Constructor of Architectural languages) wants to define new application, it searches for meta elements in the library and selects them.

2. Adapt. The CA can use the meta elements as they are or can adapt them to the needs of the application. For example in the client-server system, components must interact with each other via connectors.

3. Use. This component then can be used by the AA (Architect of Applications) to build new applications.

Architectural level

1. Search and select. The AA searches the library for types of architectural elements to define applications and gets them.

2. Adapt. The AA adapts these elements to its needs. For example in the client-server system, it could define a composite server using several components (and maybe connectors also).

3. Use. This component (server) can then be instantiated to construct an architecture.
4.3. Experimental results

Our model was used in the process of developing a customer support service application for the company Bouygues Telecom, i.e. CIS (Client Information System). The application covers three activities: service provider, distribution and sales, network provider. Each activity is represented by a branch of the model Y. We have also three different views for each activity, customer care view, billing view, and decision making view. Customer care view refers to all customer operations, e.g. development, service, follow-up, assistance, development of customer loyalty, etc. Billing view refers to the invoicing and payment services. Decision making view refers to the service of helping managers in the process of taking commercial decisions. According to the developers, the model helped significantly in the process of developing the application since we have a global, but rather riche, representation of the developed application. For confidentiality reasons we can not present the application in details in this article.

5. Conclusion and perspectives

In this article, we have presented a representation model for software architecture concepts. The model is based on the MY approach (meta modeling in Y), which is a knowledge engineering methodology. It represents software architecture concepts using three branches: component branch, connector branch, and architecture branch. The component branch represents concepts that are related to computations, the connector branch represents concepts that are related to interactions, and the architecture branch represents concepts that are related to the structure and the topology of the described system. We think that such a representation of architecture concepts aides on improving reusability not only in the implementation level, but also in the description level. The model assigns a reusable library for each conceptual level. Therefore, reusability can take place at the four levels (meta-meta level, meta level, architecture level, and application level). The idea of reusing the meta concepts is an improvement for software architecture and architecture description languages.

In this article we tested our model using a client-server system, which is an example well know in the software architecture community, next step will be experimenting the model using a real application. Another perspective is the possible integration of our model in the Rational Unified Process (RUP) [29]. RUP is a prescriptive, well-defined system development process, often used to develop systems based on object and/or component-based technologies. It is based on sound software engineering principles such as taking an iterative, requirements-driven, and architecture-centric approach to software development. As our model RUP provides several views and mechanisms, such as relatively short-term iterations with well-defined goals and go/no-go decision points at the end of each phase, to provide management visibility into the development process.
References