Towards the Convergence between Hypermedia Authoring Languages and Architecture Description Languages

Débora Christina Muchaluat-Saade, Luiz Fernando Gomes Soares
Laboratório TeleMídia – Departamento de Informática – PUC-Rio
R. Marquês de São Vicente, 225 – Gávea, 22453-900, Rio de Janeiro, RJ, Brazil
55-21-25122299
debora@telemidia.puc-rio.br, lfgs@inf.puc-rio.br

ABSTRACT
This paper presents a detailed comparison between the structural elements and definitions provided by Hypermedia Authoring Languages and Architecture Description Languages (ADL). ADLs are formal languages that can be used for representing a software architecture. Although it may look trivial to make a direct correspondence between ADL and hypermedia structural entities, such as components to nodes and connectors to links, interesting differences can be identified when observing them more closely. Based on the comparison results, a structural meta-model that can be specialized for use in both domains is proposed. Furthermore, the paper also presents an example of how the meta-model can be used for describing hypermedia document structures, showing how some features found in ADLs can be applied to hypermedia authoring languages. Our final goal is to integrate the contributions of document engineering and software architecture engineering and take advantage of the advances of one area in the other one. The current paper is the first step towards this direction.

Keywords
Hypermedia Authoring Languages, Architecture Description Languages (ADL), structural meta-model, components, connectors.

1. INTRODUCTION
Architecture Description Languages – ADL – are formal languages that can be used for representing a software architecture, defining its components, their behavior specifications, and patterns and mechanisms for interactions among them [4].

Shaw and Garlan [22] present six properties that characterize what an ideal ADL should provide:

- composition: “it should be possible to describe a system as a composition of independent components and connections”.
- Configuration: “architectural descriptions should localize the description of the system structure independently of the elements being structured. They should also support dynamic reconfiguration. As a consequence, an ADL should separate the description of composite structures from the elements in those compositions, so that we can reason about the composition as a whole”.
- Abstraction: “it should be possible to describe components and their interactions within a software architecture in a way that clearly and explicitly prescribes their abstract roles in a system”.
- Reuse: “it should be possible to reuse components, connectors and architectural patterns in different architectural descriptions, even if they were developed outside the context of the architectural system”.
- Heterogeneity: “it should be possible to combine multiple heterogeneous architectural descriptions”.
- Analysis: “it should be possible to perform rich and varied analysis of architectural descriptions”.

Analyzing these properties, we can identify some similarities with the ones that a Hypermedia Authoring Language – called HAL from here on – should provide:

- possibility of defining compositions in a document structure (composition);
- possibility of separating the description of composite elements from its constituent elements (configuration);
- possibility of describing document components and their interactions within a document (document structure) independent of the meaning of their content (abstraction);
- possibility of reusing document components inside other documents, or inside different compositions of the same document (reuse);
- possibility of combining several heterogeneous documents (heterogeneity);
- possibility of verifying the consistency (temporal, spatial, etc.) of documents (analysis).

Based on these similarities, this paper identifies common points and main differences between ADLs and HALs through a detailed comparison between software system and hypermedia document structures.
As a consequence of the comparison between ADLs and HALs, the paper also presents a precise definition of a structural meta-model that can be specialized for use when modeling software architectures or hypermedia documents. An example of how the meta-model can be used to represent hypermedia document structures is also discussed.

Our final and main goal is the formal definition of a generic model that can be used in document engineering and software architecture engineering. The idea is to integrate the contributions of both areas and take advantage of the advances of one area in the other one. For example, features found in ADLs, such as composite connectors, definition of styles, formal definition of semantics and procedures to verify and validate specifications could be applied to HALs. On the other hand, features found in HALs, such as transformation languages, modularization of languages, document adaptability and mechanisms for presenting graphical views of large structures could be applied to ADLs. Other topics, such as version control, use of XML to specify language syntax, graphical authoring tools, quality of service issues, etc. could be applied to both. In order to achieve the final goal mentioned previously, the syntactic and semantic definition of the meta-model should be provided. This paper is the first step towards it, focusing on the structural concepts.

The paper is organized as follows. Section 2 describes some important structural characteristics of ADLs, making a brief comparison among features provided by some ADLs found in the literature. Section 3 makes a detailed comparison between the structural characteristics of the mentioned ADLs and the structural characteristics of some HALs, such as NCL [3], SMIL [23] and Madeus¹ [9], highlighting their main differences. Section 4 presents the structural meta-model description that can be specialized for use in ADLs and HALs. Section 5 gives a brief example of how the meta-model can be specialized to represent hypermedia document structures. Finally, Section 6 is dedicated to conclusions and future work.

2. ADL – ARCHITECTURE DESCRIPTION LANGUAGES

An ADL can be defined as a language for specifying the high-level structure of an application, rather than the implementation details of any specific source module. It models the conceptual architecture of software systems, in spite of the system implementation. System architectural descriptions are extremely important, as they serve as a skeleton where properties can be proved showing how systems satisfy their main requirements.

In the majority of ADLs, the building blocks of an architectural description are [13]:

- components: computation or data storage units that can be as small as a unique procedure or as big as a whole application;
- connectors: used for modeling interactions among components and defining rules that govern these interactions.

Different from components, connectors may not correspond to compiled units of an implemented system. They can manifest themselves as shared variable accesses, table entries, buffers, linking instructions, procedure calls, pipes, etc.;

- architectural configurations: connected graphs of components and connectors that describe the architecture structure. This information is useful for determining if the appropriate components are connected, if their interfaces match, if connectors allow an adequate communication and if their combined semantics results in the desired behavior. Ideally, a system structure should be clearly understood from a configuration specification, without the need to study the specification of its components and connectors. Hierarchical composition is desirable in ADLs, allowing a system description in different levels of detail. A complex structure with a complex behavior may be explicitly represented or may be abstracted by a single component or connector. A whole architecture may become a unique component in a more complex architecture.

In order to infer any type of information about a software architecture, component and connector interfaces must also be modeled. A component interface is a set of interaction points with the external world. The interface specifies services (messages, operations and variables) that a component provides and services that it requests from other components. Thus, an interface defines the computational duty of a component and its usage constraints.

A connector interface is a set of interaction points that may be attached to components or other connectors, usually specifying the services the connector expects from attached elements.

We can find several proposals for ADLs in the literature, such as ACME [6], Aesop [7], CL [18], C2 [16], Darwin [11], Rapide [10], SALD [15], Unicon [21] and Wright [2]². Tables 1 and 2 summarize some of their main structural characteristics. Table rows represent the following features:

1. Composite component – specifies if the ADL provides composite components;
2. Component interface – specifies the name that component interfaces have in the ADL;
3. Dynamic configuration – specifies if the ADL provides support for dynamic components or dynamic reconfiguration;
4. First-class connector – specifies if the ADL treats connectors as first-class entities;
5. Connector interface – specifies the name that connector interfaces have in the ADL;
6. Multipoint connector – specifies if the ADL allows more than two elements to interact using the same connector;
7. Direct linking between connectors – specifies if the ADL allows a connector to be directly attached to another connector;
8. Composite connector – specifies if the ADL provides composite connectors;

¹ Madeus is a hypermedia authoring and formatting system that offers an XML-based declarative language for authoring documents. When the text refers to Madeus, it means its authoring language.

² Notice that not all the ADLs cited in this paper are considered ADLs by every author in the literature. Some of them are considered Module Interconnection Languages – MILs, for example. However, since all of them offer means for describing software architecture structure, they are going to be called ADLs in this paper.
Constraints on mappings between interfaces of nested elements – specifies if the ADL has any constraint on how to map composition interface points to their constituent interface points;

10. Architectural style definition – specifies if the ADL provides the definition of styles, which describe families of software systems, instead of just a specific one;

11. Constraint definition – specifies if the ADL allows the definition of constraints among architectural elements;

12. Several representations for composite elements – specifies if the ADL allows the definition of different lower-level representations for the same composite element.

Table 1. Summary of structural characteristics of some ADLs

<table>
<thead>
<tr>
<th>ADL</th>
<th>ACME</th>
<th>Aesop</th>
<th>CL</th>
<th>C2</th>
<th>Darwin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>port</td>
<td>input port</td>
<td>entry port</td>
<td>provided port</td>
<td>provide service</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>role</td>
<td>role</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>11</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2. Summary of structural characteristics of some ADLs

<table>
<thead>
<tr>
<th>ADL</th>
<th>Rapide</th>
<th>SADL</th>
<th>UniCon</th>
<th>Wright</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>provides requires in/out action services</td>
<td>iport</td>
<td>oport</td>
<td>player</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>role</td>
<td>role</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

In order to specify component behavior and interaction protocols among components, ADLs usually provide a semantic description of the system, using a formal language like CSP, π-calculus, Petri nets, etc., or even using a meta-language for specifying semantics [27]. However, this paper only focuses on the structural definitions of ADLs and HALs, without considering how semantics are specified.

3. COMPARING ADLs TO HALs

At first glance, it may look trivial to make a direct correspondence between the basic structural elements found in ADLs and the ones found in HALs. Components correspond to nodes, connectors to links and architectural configurations to composite nodes.

The hierarchical description of a software system is analogous to a hierarchy of nested composite nodes in a hypermedia document, with the difference that, in a software system, the graph of constituents (components and connectors) must be connected, while this is not required in a hypermedia document.

However, observing more deeply, we should note that the correspondence is not so direct and trivial. There are some interesting structural characteristics of ADLs that have no analogy in HALs, and vice-versa. This section aims at highlighting and discussing these characteristics, creating a start point for defining a generic structural model that can be used in both domains.

### 3.1 Component Interface

ADLs define the component interface as a set of interaction points, usually called ports. Moreover, some ADLs classify the interaction points in the set, establishing a type for each one. Examples are entry and exit ports in CL, provided and required ports in C2, provided and required services in Darwin and iport and oport in SADL. Besides this basic classification representing the direction in which a component will interact when using the interface point, there are other criteria for specifying types. For example in Rapide, component interface elements may be provides, requires, in and out actions and also services, depending on the type of interaction that a specific interface point may participate.

In HALs, a node interface is given by its list of anchors, and usually an anchor represents a marked region in the content of a node used in a link definition. So far, there had been no need for distinguishing different types of anchors, as any of them can be source or target endpoints of links or can participate in any kind of relationship among nodes. However, HALs will also need another type of node interface point for their composite nodes, but we have to postpone this discussion to Section 5.

### 3.2 Connectors

Making a correspondence between software connectors and links in a hypermedia document is not trivial at all. Connectors have a great importance in software architecture descriptions, and the support for modeling connectors provided by ADLs sometimes is similar to the support for modeling components. Indeed, the main differences between ADLs and HALs are related to connectors, as discussed in what follows.

#### 3.2.1 Connector Cardinality

Some ADLs, such as ACME, Aesop, C2, UniCon and Wright, allow more than two components to interact using the same connector, while others, such as CL and Darwin, do not. The same scenario is found regarding HALs. Some of them do not allow n-ary relationships among nodes represented by links, such as HTML, SMIL and Madeus, while others do, such as NCL, XLink [29] and Petri-net-based HALs like Trellis [26] and I-HTSPN [28].

#### 3.2.2 Connectors as First-class Entities

Some ADLs model connectors explicitly, clearly distinguishing connector types from connector instances. In this case, connectors
are considered first-class entities [13, 20], that is, they can be named, subtyped and reused. Other ADLs always specify connectors as instances, such as Darwin and CL, preventing them to be manipulated during design or reused in the future. In this case, they are called in-line connectors.

A first-class connector can be defined independently of components or configurations where it will be used. Usually a configuration defines instances of component and connector types, also specifying which connector instance interface points are attached to which component instance interface points. ACME, Aesop, C2, SADL, UniCon and Wright are examples of ADLs that treat connectors as first-class entities.

In some HALs, a hypermedia node can be reused in different compositions. On the other hand, hypermedia relations expressed by links cannot be reused in the majority of HALs. In some HALs, links are embedded in the node content, preventing the node reuse without bringing the link together, such as in HTML. In some others, links are contained in link sets, which are defined in composite nodes, as in NCL. In this case, media nodes or composite nodes can be reused, the link set alone cannot. Link sets can also be stored in independent repositories, as in the link bases of Microcosm [5] and XLink, for example. In this case, the same set of documents can be associated to different link sets, but the relation expressed by a link cannot be reused yet. Other HALs provide predefined sets of relation types, such as Madeus high-level constraints and the typed transitions of I-HTSPN, for example. In this case, relation types can be used to create links in any document. Although these HALs provide the concept of relation type and relation instance, they do not provide support for the definition of user-defined relation types. For all these reasons, relations expressed by links in HALs do not have the same treatment connectors do in ADLs.

In order to give first-class status to relations expressed by links, a link specification has to be divided in two elements. The first one defines the relation and the other defines the connections between the first element and the components (hyperdocument nodes) participating in the relationship. Some HALs already do this division. In NCL, for example, the first element of a link is called meeting point and the second the endpoint set. However, NCL does not allow a meeting point to be reused and it treats both link elements as link embedded attributes.

3.2.3 Connector Interface

The majority of ADLs that treat connectors as first-class entities allow the definition of connector interfaces. The duties of components participating in a connector interaction are specified by a set of elements of the connector interface, usually called roles, as in ACME, Aesop, UniCon and Wright.

When a configuration defines which connectors are attached to which elements, in fact it specifies bindings between a component interface point (port) and a connector interface point (role). These binding elements are called attachments in ACME and Wright, communication links in C2 and ducts in [12]. In ADLs that provide in-line connectors, no connector interface is offered, and a connector directly binds two component interface points.

ADLs do not classify connector interface points using types, as some of them do with component interface points. For example, they do not define which roles are source or target of an interaction. In fact, if there is such sense of direction, it is defined by the component port type to which a role is attached.

Since relations defined by links have not been treated so far as fully first-class entities in hypermedia conceptual models, no HAL explicitly defines the link relation interface. In HALs that provide predefined relation types and separate relation types from relation instances, the relation interface is implicitly predefined by the relation type. For example, in Petri-net-based HALs, transition types represent relation types, and arcs arriving at or leaving a transition implicitly define the role of each interacting node, which is represented by a place.

Hypermedia relations expressed by links may have causal or constraint semantics. When representing constraint relations, no sense of direction is present in a link. When representing causal relations, conditions relating link source endpoints have to be satisfied in order to fire actions relating link target endpoints. In this case, defining the direction of the interaction is essential. Since node anchors may be source or target of any link, node interface points are not typed for this reason. Thus, relation interface points will certainly need types to express causality, as will be discussed in Section 5.

3.2.4 Direct Linking between Connectors

Although it is not an ADL common feature, C2 allows a connector to be directly connected to another connector [16].

This feature is not allowed in the majority of HALs either, where a link must necessarily connect nodes, although link anchoring on another link was allowed once in the MHEG [14] initial proposal.

Whether or not this feature should be considered in a generic structural model for HALs and ADLs is a point that remains to be discussed.

3.2.5 Composite Connectors

One of the main differences between HALs and ADLs is the possibility of defining composite connectors in a software architecture.

Composite connectors, called high-order connectors by Garlan [8] and discussed comprehensively in [12], represent compositions of several connectors and components, modeling more complex interactions between components of a software architecture.

Therefore, the hierarchical description of a software architecture can be done encapsulating subsystems as components as well as connectors. It is worth mentioning the discussion found in the literature related to whether connectors should be distinguished from components. There are ADLs that make no distinction, modeling complex connectors as components, such as Rapide [10] that provides a connector component. However, this approach is criticized by other works [1, 12]. They argue that the basic role of a language is providing an expression vehicle that matches the user intuition and practice. Thus, if abstractions have distinct semantics, they must be represented by distinct language elements.

Note that node interface points may be typed for other reasons, as already mentioned in Section 3.1.
Several works discuss the use of composite connectors [2, 6, 7, 8, 12]. ACME, Aesop and Wright offer mechanisms for defining composite connectors and clearly distinguish them from composite components.

As far as we know, HALs do not have the concept of composite relations expressed by links, although some works about hypermedia versioning [24] mentioned that link versioning becomes necessary when complex relations are expressed using links.

In a generic structural model, composite connectors must certainly be provided.

### 3.3 Mappings between Interfaces of Nested Elements

When it is not possible to create a direct connection between an element inside a composite element and an element outside the composition, a mapping between interface points of nested elements is needed, as illustrated in Figure 1.

![Figure 1. Definition of interaction between components in distinct configurations](image)

Some ADLs make constraints concerning that mapping, requiring that the constituent element type match the composite element type. Examples of those constrained mappings are the rep-maps of ACME and the bindings of Aesop and Wright. Other languages, such as C2, do not make any constraint concerning this mapping.

Still in other ADLs, the relation between nested element interfaces is represented by an in-line connector (see Section 3.2.2) attaching parent and child components, as the binds of Darwin and the links of CL.

Some HALs allow a direct connection between nodes in different compositions, such as NCL, as illustrated in Figure 2(a). In this case, when node reuse is also offered, a link must distinguish to which nested node it refers. For example, the link shown in Figure 2(a) touches node B inside node F instead of touching the occurrence of B inside composite node D. This distinction is made by identifying the path of nested compositions used to reach a link anchor node. That path is analogous to a set of mappings between a composition and its constituent in the path, as shown in Figure 2(b). However, NCL treats this path as part of the link endpoint definition and does not provide the concept of mapping.

![Figure 2. Link between nodes contained in distinct compositions](image)

### 3.4 Several Representations for a Composite Element

In some ADLs, composite elements may have more than one representation, which are different detailed lower-level descriptions. The advantage of allowing multiple refinements for the same element, no matter if it is a component or a connector, is to provide different views of the same architectural entity. However, there is no way of making any correspondence among the different views, such as specifying which component of a representation corresponds to which component of another representation.

Some HALs provide means for specifying different presentation specifications for the same node, which may be composite or not. This feature allows the same document node to have presentation alternatives that may be chosen depending on the navigation made to reach the node. In [25], the alternatives are called node representation versions and they are treated as versions of the same node.

Other HALs provide a way of grouping different alternatives for a document node through a special type of composite node, which is called switch element in SMIL and query composite node in NCL. This special entity allows defining a selection criterion, where only one of its components (as in SMIL) or a subset of them (as in NCL) should be selected during runtime.

Those features provided by HALs may be compared to the ADL feature of offering different representations for the same architectural element. Similar to ADLs, HALs provide no means for making a correspondence among different representations of the same composite element. This is an interesting topic for future work with contributions for both areas.

### 3.5 Architectural Styles and Constraints

ADL styles are useful to represent well-known structures that are frequently used in several applications and projects. An architectural style specifies a family of software systems, while a configuration defines a specific system. A style is composed at least by two distinct parts [22]:

- a vocabulary of design elements – component and connector types, such as pipes, filters, clients, servers, parsers, databases, etc.
- A set of configuration rules – or topological constraints – determining the permitted compositions of those elements. For example, the rules might forbid cycles in a pipe-and-filter style, specify that a client-server organization should be an n-to-one relationship, etc.

ADLs frequently provide the possibility of establishing constraints associated to architectural styles, obliging all systems that follow a certain style to satisfy the constraints the style specifies. Examples are Aesop style configuration rules and Wright style constraints. On the other hand, other ADLs, such as SADL and Rapide, allow the definition of constraints inside any configuration.

The W3C XML Schema [31] Recommendation allows the definition of XML document types and also constraints on document elements. Examples of constraints are specifying the minimum and maximum number of element occurrences in a...
works studying the possibility of applying document authoring
capabilities to architectural styles in ADLs. The model is a special case of compound graph with two types of
ergies, called component and connector, each one having a set
of interface points.

### 3.6 Dynamic Configuration

Darwin provides the definition of components that may be
dynamically created or removed during document navigation. For example, a client-server system may be specified as a static
server module plus client modules that may be dynamically
created at runtime.

Although CL does not offer dynamic components, it provides
dynamic reconfiguration for software modules. A CL designer
may define some modifications and consider them as possible to
documents. For example, those modifications may be removing a
module from the system, destroying an instance of a module or
destroying interactions between two component interface points.

The model is a special case of compound graph with two types of
vertices, called component and connector, each one having a set
of interface points.

The model also defines a basic type of arc, called bind, connecting
a component interface point to a connector interface point. Binds
must not directly connect component to component or connector
to connector. If such a connection is necessary, intermediate
dummy vertices (components or connectors) may be created in
order to represent it. Note that in all ADLs that provide in-line
connectors, a component-to-component connection will have to
be represented by a dummy connector attaching those components. Analogously, in C2, a dummy component will have
to be used for representing direct connections between connectors.

A simple structure defined by the meta-model is graphically illustrated in Figure 3.

![Figure 3. Example of a simple structure](image)

### 3.7 Using XML for Describing Architectures

Several ADLs are specified using XML [30]. There are current
works studying the possibility of applying document authoring
capabilities to architectural styles in ADLs. The model is a special case of compound graph with two types of
vertices, called component and connector, each one having a set
of interface points.

The model also defines a basic type of arc, called bind, connecting
a component interface point to a connector interface point. Binds
must not directly connect component to component or connector
to connector. If such a connection is necessary, intermediate
dummy vertices (components or connectors) may be created in
order to represent it. Note that in all ADLs that provide in-line
connectors, a component-to-component connection will have to
be represented by a dummy connector attaching those components. Analogously, in C2, a dummy component will have
to be used for representing direct connections between connectors.

A simple structure defined by the meta-model is graphically illustrated in Figure 3.

![Figure 3. Example of a simple structure](image)

Like in a compound graph, vertices may represent subgraphs also
composed by components, connectors and binds. Moreover, compositions may be nested to any depth.

A composite component may be directly connected, through a
bind, to a constituent connector. This will be necessary to model
interactions between parent and child components allowed in
HALs. On the other hand, a composite connector is not allowed to
be directly connected to a constituent component (these
interactions are neither allowed in ADLs nor in HALs). Figure 4
illustrates composite vertices.

ADLs and some HALs allow vertices inside a composition to be
connected to vertices outside. These connections are always
indirectly made in ADLs and can be directly or indirectly made in
HALs. Without loss of generality, as we shall see, direct
connections are not allowed in the meta-model, since binds can
only be defined between vertices having the same parent or
between parent and child vertices.

In order to allow indirect connections between vertices inside a
composition and vertices outside, another type of arc is needed,
called map. A map connects parent and child vertices of the same
type, such as a parent component/connector interface point to a
child component/connector interface point. Connections between
vertices inside and vertices outside a composition can be done
through arc paths containing binds and maps. Figure 5 illustrates
the use of maps. It is important to note that, different from binds,
maps do not represent interactions between components or
between connectors. They are only means for exporting internal
interface points to the outside of a composite vertex.

---

4 Only C2 does not make constraint concerning the mapping
between interfaces of nested elements, but all other ADLs do.
This constraint was maintained in the meta-model.
It is important to maintain anchors for hypermedia composite
node, an anchor may be any subset of its constituents. A
information unit of the node content. An information unit
depends on the media node type and can be a character for a text
node without loosing the property of compositionality. The use of
composite node ports will allow, among other things, the
presentation of its constituents. Through the use of meta-model
maps, a composite node can make an interface point of a
constituent node visible for any relation with any node outside the
composite node.

Composite nodes must then have two different interface point
types, anchors and ports, justifying the statement previously
mentioned in Section 3.1.

Our hyHAL connector represents a spatio-temporal relation that
will be used in the creation of links among nodes. It specifies the
relation semantics, but does not specify node anchors that will
participate in the relationship. Node anchors will be specified by
links that use the connector. Moreover, considering that our
hyHAL is not restricted to a predefined set of connectors, but
provides the definition of new connectors by users, it indeed treats
connectors as fully first-class entities, just like ADLs do.
Connector interface points are called roles. Special types of roles have to be defined for different types of relations, as mentioned in Section 3.2.3. In the case of causal hypermedia relations, role types may be conditions and actions, and in the case of constraint hypermedia relations, role types are always conditions.

In order to specify how participants interact, a special connector attribute called glue [2] is needed. In a causal hypermedia relation, the glue defines an expression relating condition roles, which must be satisfied to fire another expression relating action roles, also defined in the glue. In a constraint hypermedia relation, the glue only describes the constraint expression relating condition roles.

Note that the semantic description specifying the role and glue behaviors is extremely important, as it gives the semantics of a hypermedia document presentation. Since this paper is only concerned about characteristics of the structural model, the hyHAL semantic description is neglected. In [17], a detailed presentation of role and glue semantics is given.

A link references a hypermedia connector, and also defines a set of binds relating the connector roles to node interface points, which may be anchors or ports. If the node interface point specified in a bind is a composite node port, there must be a map path relating that port to constituent node interface points, until an anchor is reached. The complete definition of a link is obtained by:

- a reference to a hypermedia connector;
- a set of binds relating the connector roles to node interface points;
- a set of map paths, in the case of binds to composite node ports, relating these ports to internal node interface points, defining sequences of nested composite nodes until an anchor of the most internal node is reached in each sequence.

Note that, in our hyHAL, it is possible to reuse the definition of spatio-temporal relations (represented by connectors) in several different links, that is, links relating different nodes, but having the same causal or constraint semantics. For example, a document represented by composite node COMP2 in Figure 7 defines three different links using the causal hypermedia connectors CON1 and CON2 illustrated in Figure 6. Connector CON1 specifies a traditional hypermedia relation having the same HTML link semantics, where a selection in a node anchor, described by the selection_condition role, fires the presentation of another node anchor, represented by the presentation_action role. CON1 was used to create links 11 and 12 inside document COMP1, through four different binds. Connector CON2 specifies an n-ary hypermedia relation, where a selection in a node anchor, described by the selection_condition role, happening during the presentation of another node anchor, described by the presentation_condition role, fires the presentation of a third node anchor, described by the presentation_action role. CON2 was used to create link 13 inside document COMP2, specifying that a selection in a specific anchor of node B, during the presentation of media node D, fires the presentation of node E. Note that, in order to make anchor k of node B play the selection_condition role of CON2, link 13 must have defined a bind (COMP1, i, CON2, selection_condition), where i represents a port of COMP1. Furthermore, a map (COMP1, i, B, k) also has to be defined, making port i of COMP1 export anchor k of node B.

![Figure 6. Examples of causal hypermedia connector](image)

![Figure 7. Example of hypermedia document](image)

![Figure 8. Example of a composite hypermedia connector](image)

![Figure 9. Example of hypermedia document](image)

6. FINAL REMARKS

Integrating the results already achieved in the areas of document engineering and software architecture engineering and to take advantage of the advances of one area in the other one is the final goal of our work. The first step in this direction has been the definition of a generic model that can be used in both domains.

---

5 Other types of node interface points may also be used in HALs. For example, node attributes may be considered node interface points, since they may be needed for specifying spatial relations [17].
Features found in ADLs, such as giving connectors first-class status, have already been applied to HALs, as described in [17], showing that the final goal may be achieved.

In this paper structural definitions provided by ADLs and HALs were studied and compared and a generic structural model was proposed. An example of how the meta-model concepts could be applied to a hypothetical HAL was also discussed. As aforementioned, the meta-model semantic definition was left for future work.

Although the explanation about the specialization of the meta-model for the hypermedia domain was only briefly made in Section 5, this was an important contribution to hypermedia models, as explained in details in [17]. Some advantages found in ADLs were brought to HALs, such as reusing a spatio-temporal relation specification in different links with the same behavior, and providing composite spatio-temporal relations expressed by links, which is a new feature for HALs.

It remains to be studied, how composite nodes with special hypermedia semantics, such as parallel and sequential compositions, could be represented with the meta-model. Maybe a generalization in the use of styles would answer this question. Styles allow the definition of constraints on the behavior of components and connectors inside a configuration. Applied to hypermedia, they would allow the definition of document structures with particular characteristics to be reused in many different documents.

As previously mentioned in the paper, studying how to make a correspondence among different representations of the same composite element is a future work that can contribute to both ADLs and HALs. Another research topic is to study the correspondence between ADL support for dynamic configuration and HAL support for virtual entities.

Another important future work is to make solutions proposed for problems found in the hypermedia domain available for the software engineering domain. One example is related to the presentation of graphical views of complex and large structures. Another example is the support for managing the evolution of a structure over time through the use of version control mechanisms, considering different representations for the same component as versions. A third example is the use of notification mechanisms when cooperative authoring is needed. Solutions for those problems have already been proposed for hypermedia documents and if they are generalized for the meta-model entities, they can be directly applied to software architecture descriptions.

7. REFERENCES


[17] D.C. Muchalut-Saade, L.F.G. Soares. Hypermedia Spatio-Temporal Synchronization Relations also Deserve First-


