A specification schema for software connectors
Slotos, Thomas

Published in:
A specification schema for software connectors

DOI:
10.1145/2602458.2602464

Publication date:
2014

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
A Specification Schema for Software Connectors
Thomas Slotos
Leuphana University of Lueneburg
Scharnhorststr. 1
21335 Lüneburg, Germany
++49 4131 677 2196
slotos@leuphana.de

ABSTRACT
Since the invention of the connectors that serve as one of the building blocks of software architecture, various researchers have described the characteristics of connectors that they have discovered in given software systems. These characteristics have frequently been revealed by employing a bottom-up approach to different disciplines of software engineering. As a consequence, numerous attributes of connectors have been introduced at a rather technical level, and the vocabulary regarding specific aspects of connectors is highly heterogeneous. This eclectic list of attributes is not an appropriate abstraction level for an architect when developing connectors for application software. Instead, it would be more helpful to have a framework that begins from a conceptual perspective and guides the software architect in identifying the attributes required for a specific connector. To this end, a hierarchical schema for specifying software connectors is proposed in this paper. In contrast to existing schemas in which either the attributes considered are few in number and ungrouped or the attribute specification and grouping was not the main focus of the research, this paper presents a homogenous specification schema at different levels of abstraction and is based on logical perspectives rather than technical features. The schema can support architects in various situations—such as connector selection and documentation—and can aid in implementing or generating concrete software connectors. The application of the schema will be demonstrated by specifying a software connector between Twitter and an application.

Categories and Subject Descriptors
D.2.11 [Software Architectures]

Keywords
Connector; specification; software architecture.

1. INTRODUCTION
After realising that software products were becoming increasingly complex, there was a need to address this phenomenon. One common approach is to investigate systems and extract commonalities found in different types of software. These structural commonalities are called “architectural styles or patterns” and are based on concepts and connectors. As one of the results of defining architectural styles, connectors were introduced as “first class citizens” by Shaw [33]. There are diverse definitions of software connectors that describe their different features [34][2]. One of the most succinct among these is that “Connectors are functional bindings between components enabling their controlled interoperability” [26]. Since the introduction of connectors to software architecture, several research questions regarding their description, categorisation and development have arisen. As a basis for answering these questions, studies from different areas of software engineering have explored existing systems that employ these considerations. In the absence of a general concept, a common approach has been to characterise the qualities of software connectors primarily at a technical level. Insights from given technical instances of connectors are abstracted to a technology-independent level using a “bottom-up” approach. Step by step, characteristic properties of connectors have thus been revealed. However, these properties are distributed over numerous research papers in diverse scientific fields and at a variety of levels of software engineering. A heterogeneous vocabulary has thus emerged; as a consequence, efforts have been made to collect and group these properties in schemas that are based primarily on the technical origins of connectors found in existing systems [34][13]. For an architect who must map architectural constraints in the form of high-level requirements for concrete connector attributes, there is a mismatch of different abstraction levels. To overcome this mismatch, I argue that a specification schema is required that integrates these opposing views into a unified form.

The need for such a view has also been noted by Hirsch et al.: “If connectors could be characterised by means of high level properties, descriptions would be much more declarative and adequate for architectural analysis” [13]. The need to distinguish “what” should be generally supported by a connector from “how” these described attributes should be accomplished with a certain implementation at a (technically) lower level stands behind the idea of a high-level specification schema for software connectors. Hirsch et al. [13] state that “… working with high level properties can deliver architectural mechanisms that are not bound by language features”. This concept of different types of views is not new to software architecture [27]. For example, Hofmeister et al. distinguish among four different views: conceptual, module, code and execution [14].

In this paper, a high-level view of connector attributes is presented that can be assigned to the conceptual view of software architecture. The schema collects characteristic attributes of software connectors from various research areas and groups these attributes into categories at different levels of abstraction, forming a unifying specification structure that considers not only research papers from the component-and-connector approach but also results from such areas as distributed systems [37], middleware [17], architecture description languages [28] and enterprise application integration [15].
The remainder of the paper is organised as follows. Section 2 consists of a brief review of the literature. Section 3 addresses the unified specification schema for software connectors. Section 4 describes how the specification schema can aid a software architect and the purposes for which it can be used, which thereby illustrate the benefits of such a schema. In Section 5, the specification schema is applied to a concrete example to illustrate its use in practice. Section 6 concludes.

2. RELATED WORK

Since the introduction of the concept of a “software connector”, researchers have worked on describing the characteristics of connectors by establishing different types of specification dimensions and attributes. The study from Shaw and Clements [34] was one of the first to introduce the dimensions of data and control flow. In their research, they examined the characteristics of connectors and stated that “The major axes of classification are the control and data interactions among components”. Certain other attributes of data and control flow were also mentioned, but the main focus of their study was to establish a “uniform descriptive standard for architectural styles”.

One of the next important contributions was the idea to formulate a “canonical set of primitives [that] would provide a framework for comparing, refining and reusing connectors” [13]. The first step towards such an organisation of attributes was a tabular schema presented by Hirsch et al. [13] that included a list of connector properties that are applied to differentiate connector types. Because connector classification was the focus, the schema lacks any grouping of attributes at different levels of abstraction and the collection of attributes is rudimentary. A similarly organised specification schema is presented in [10].

The most important improvement towards a unifying specification schema was achieved by Mehta et al. [29], whose main contribution was to distinguish four general services (communication, conversion, coordination and facilitation service) and to differentiate among eight connector types for interaction (procedure call, event, data access, linkage, stream, arbitrator, adaptor and distributor). Characterisation criteria at different levels are used to define specific connector types. Because Mehta et al. aimed to provide a taxonomy for connectors, the classification begins with connector types at the root and results in connector properties that are assigned more than once to a connector type. Their intentional decision for that type of entry point into the specification structure leads to a classification that “does not result in a strict hierarchy, but rather in a directed acyclic graph (DAG)” [29]. The specification schema presented in this paper is a strict hierarchy that is completely independent of any technically motivated entry point at the beginning; this decision was taken because of a broader view about connectors that incorporates more than the interaction mechanisms of connector types that were the formative force for Mehta et al. To achieve such a broader view, it is necessary to address connectors at the conceptual level and to be guided by general principles that are not technically driven. In contrast to Mehta et al., the approach to specifying software connectors in the present paper has narrowed its focus to the design phase of software development. The subsequent phases of coding, compilation, linkage, deployment and runtime—in which connectors are also relevant—are not covered because the schema is designed for software architects and not for application programmers differentiating “what” should be specified from “how” these attributes could be realised.

The two approaches do not conflict but complement one another. In principle, combining the two schemas is possible over the described dimensions with respect to the attributes they have in common.

3. THE SPECIFICATION SCHEMA

There are different types of structuring possibilities for specifying software connectors depending on the task and background of the creator. Assuming that such a specification schema is used by architects who must map the non-functional requirements of a future system to the concrete attributes of connectors within that system, beginning from a conceptual level would minimise the complexity of the specification process. Therefore, in this paper, an elaborated specification schema is presented that describes software connectors from both an external view and an internal view.

From an outside perspective on the connector, the internals within the connector are hidden, and only essential attributes that are required by the components that the connector interconnects with are to be considered. This type of perspective will be called the “external view” (Figure 1). For example, two distributed components in a software system lead to the construction of a connector that supports remote communication. Or, a uni-directional connector between two components is only necessary if one component is dependent on the functions of the other component, but not the reverse. After describing the attributes that are enforced by the components, the inner properties of the connector that are mostly independent of the components should be specified in the “internal view” (Figure 2). The inner set-up of a connector is based on decisions that relate to the tasks of the connector (e.g., transformation and filtering) or control flow considerations (e.g., pull vs. push mechanism).

Both the external and the internal views are successively refined such that similar attributes are grouped together at different levels, which are called “dimensions” and “criteria”. At the end of this specification hierarchy, “attributes” or “sub-attributes” terminate the description structure. For each attribute, concrete values are given from which an architect can choose characteristics for a connector type or instance. These values are suggestions that are extracted from various research papers and integrated in the schema. The proposed values neither form a complete set nor constitute the only possibilities for specifying software connectors. Other specification forms, such as diagrams, can also support the description of connector aspects (e.g., the composition of a connector might be described by a class diagram).

3.1 External View – dimensions, attributes and values

To describe a connector from the external view, several criteria for the dimensions space and time should be specified.

3.1.1 Space dimension

The space dimension consists of the criteria location, cardinality and directionality.

A connector can be either local or remote, depending on the location of the connecting components [15].

One of the basic tasks of a connector is coordination among components. How many components are the subject of coordination is indicated by the cardinality at each end of the connector. Only three types (1:1, 1:N and M:N) of quantifying relations between components are necessary to characterise a connector. In software architecture, a 1:1-relationship is known as
a unicast connection and a 1:M-relation as a multicast or broadcast communication [34], [5].

The directionality of a connector indicates whether components at both ends are invoked (bi-directional) or if it is only necessary to call the functions of a component at one end of the connector (uni-directional) [34], [24].

3.1.2 Time dimension
An external perspective on the time dimension of the connector includes criteria such as communication, processing mode and durability.

Selecting one of the values of the communication criterion for a connector determines how data will be transferred with regard to time considerations. In general, it is possible to differentiate between a time independent (discrete) and a time dependent (continuous) mode of data transmission. Using time independent communication indicates that there are no constraints from the receiver regarding the transmission time for data. For time dependent communication, the target component has a communication constraint on the connector to deliver the data continuously (e.g., a video data stream) [37]. The continuous transmission of data is specified by the isochronous value.

Discrete communication can be further subdivided into the two attributes synchronisation and provisioning. Synchronisation is coordinating data between a sending and a receiving component by a connector. The strength of the coupling (loose or strong) between components after one component has initiated a call to the other component via a connector determines whether there is a synchronous or an asynchronous invocation [15]. A call is considered synchronous if it blocks the caller until the caller receives a response. By contrast, if the caller is not blocked after sending data to the target component, there is an asynchronous communication between the sending and receiving components. A synchronous call can be understood as a strong coupling between the sending and receiving component because the execution of the sender is blocked until the response is delivered. Using asynchronous communication allows the sending component to execute further statements (non-blocking behaviour) and to receive the response to the call later in time.

A determination of the provisioning of the data in the connector is required when the data cannot be delivered immediately because the receiving component is not accessible. In using persistent communication, the connector stores the data as long as necessary to deliver it to the target component. By contrast, transient communication is applied when the connector discards the data because the receiving components are not online [37].

Connecting at least two components is one of the main tasks of a connector. In specifying the durability of a connector, the software architect selects either connectionless (non-durable) or a connection-oriented (durable) communication between components [13].

Depending on the sending or receiving time constraints of the components, the architect must design a connector that uses either sequential or parallel communication paths [34]. In sequential communication, the data are transported one after another. Using parallel communication for data transportation instead is faster because several pathways for data delivery are available.

3.2 Internal view – dimensions, criteria, attributes and values
Mehta et al. indicate that a “connector can also have an internal architecture ...” [29]. The internal view of a connector can be differentiated according to the dimensions structure, behaviour and information. Clements et al. [9] offer the following definition: “A software architecture for a system is the structure or structures of the system, which comprise elements, their externally visible behaviour, and the relationships among them.” Because software systems manipulate and transport information within the system and to neighbouring systems, the third important dimension of a connector is information.

3.2.1 Structure dimension
Two criteria comprise the structure dimension: horizontal and vertical. The horizontal perspective describes the component set-up that sketches how many inner functional units there are and how they are arranged “At the architecture level ... capturing connectors as a composition of connector elements, which can be primitive or composed” [7] A connector is called atomic if it consists of a single building block that is sufficient to satisfy the constraints on the connector. For example, Shaw and Clements [34] indicate that procedure calls might be understood as a type of undividable connector. More common are composed connectors, which are made up of several chained functional units that represent the connector architecture [36], [25], [23].

Figure 1. External view
The vertical perspective on the structure of a connector describes how the functionality inside the connector is based on other services that are provided by lower layers. As with the ISO/OSI [39] model for specifying interconnection among physical/logical computation units depicted in a layered design, the architecture of a software connector can also be based on layers of different abstractions. Instead of implementing required functionality (e.g., protocol adaptation) on its own, a software connector can utilise a service provided by the platform. Middleware as a common interconnection platform facilitates several general services, which can be included in the connector design leading to a layered connector architecture [17]. If layered is selected, a more detailed description of the underlying services within a specification is often necessary.

3.2.2 Behaviour dimension

In the behaviour dimension of the internal view of a connector, the order of the data transmission, often called control flow, is considered. It is important to note that data, which are transferred according to a control flow definition, are not changed by the connector [34], [29].

To specify the control flow of a connector, several attributes are necessary to guarantee the correct ordering of the data. To group these attributes, two criteria are established for the schema, called endpoint-oriented and data-oriented behaviour. Whereas the endpoint-oriented criterion represents attributes for handling the connection to the components, the data-oriented criterion includes attributes that add functionality while data are transferred through the connector. Interaction, content transfer strategy, invocation mode and protocol adaptation are the attributes that belong to the endpoint-oriented criterion. The data-oriented criterion encompasses the transaction, security, loading strategy, buffering, paging and filtering attributes.

Next, the attributes of the endpoint-oriented criterion are described. For interaction requirements, the software architect chooses between an “exogenous” and an “endogenous” connector (see [22]). An exogenous connector controls the coordination between components, whereas this task is assigned to the interconnecting components with an endogenous connector.

The content transfer strategy of a connector indicates how data are received from or sent to a component and can be chosen from two different strategies. In using the pull strategy, the connector fetches data from the component, which indicates that the connector determines the point in time when data are required. A connector uses the other mechanism (push) when it initiates data delivery to a component [7].

The manner in which a source component delivers data is determined by the invocation mode. A connector uses an implicit or event-based mechanism if the target components are anonymous for the source component. By contrast, explicit method-invocation is used when the concrete type of the target components must be known by the source component [12].

Sometimes a connector must support different protocols at its endpoints, such as when the connecting components provide
different protocols, which is known in the connector community by the term “protocol adaptation” (see [2]).

Next, the data-oriented criterion and its attributes are considered. The attributes of this criterion regulate which data are transferred through the connector and how rapidly, securely and consistently this transfer is. These attributes can be added between the endpoint-oriented attributes. This added functionality is called the “aspect” or “crosscutting concerns” in aspect-oriented programming (AOP) [19], and Batista et al. call a connector that uses aspects “aspectsual” [3]. These authors state that “the presence of crosscutting concerns in an architecture design leads to new aspectual connectors, but interfaces and components remain the same” [4].

One of the crosscutting concerns regarding data is the notion of transaction. Adding a transaction to the connector leads to the opening of a transaction after data enter the connector and the completion of the transaction after the data is delivered to the target component [7]. If transactional is defined, then more refined transactional properties (e.g., whether the same transaction or a new transaction is used) can be described in the extended specification for that attribute.

Similarly, a secured connector can be achieved by adding a security mechanism such as authentication to the transmission functionality of the connector [32].

During the flow of data through the connector, filtering operations for the data can be applied. Whether and what type of filtering is enabled should be part of the functional specification of the connector.

The next three attributes address how data are provisioned within the connector. Provisioning is required because of the different memory sizes and processing speeds of the interconnecting components. If a loading strategy inside a connector is applied, it can be differentiated between eager and lazy loading. Eager loading is used if data dependent on the requested data are loaded simultaneously. Lazy loading is enabled when the dependent data are loaded by a later request [20], [21]. When data can be transferred depends on the availability of the components and its processing speed. If the initiating component delivers the data faster than it can be received from the target components, some type of buffering is required [15], [25]. If both components are available and have established a synchronised processing speed, direct data transport occurs. A specialised strategy for data buffering is paging. The paging attribute indicates that a subset of a result set will be organised in blocks and held by the connector [6]. When the requesting component requires the next block of data (page), the paging mechanism determines the appropriate range of data within the subset, which is then delivered by the connector. This type of provisioning facility is often called pre-fetching [8].

3.2.3 Information dimension
The information dimension of the connector describes the attributes for the data that must be transferred to and from a component and the functions that will be executed on the data while they are transported through the connector. These attributes are often called data flow [34], [29].

The composition, payload type and volume attributes are relevant for the “data” criterion.

How the data are structured is defined by the composition attribute, which is represented by the two values, “atomic” and “composed” [15]. Data consisting of many parts are considered “composed”, whereas logically indivisible data are “atomic”.

There are two alternatives for data representation for a payload type: textual and binary. In a textual representation, the content is readable by humans as well as by machines. Binary representation is used for data when the content must be in a format on which machines can operate (e.g., transferring objects).

The volume attribute indicates how much information is conducted through the connector. There are two possible extremes for this attribute: low-volume and high-volume [34]. These values represent a rather general description, and leave specifying the concrete amount of the data volume to the architect.

The specification of the function criterion describes what type of processes operates on the transferring data. Preparation and representation transformation are the attributes that should be considered. In contrast to the data-oriented criterion in the behaviour (control flow) dimension, whose attributes do not change the data while they are transferred through the connector, the functions of the information (data flow) dimension manipulate data while transmitting them from component to component.

If the preparation of data is active, then the attribute values “splitting” or “merging”, “enrichment” or “reduction” and “content-to-content translation” must be considered at minimum as modifying functions on the data [36], [15]. Splitting occurs when the original data are divided into smaller parts due to the properties of the content or meta information. The opposite operation occurs when two or more data parts are merged according to a criterion. Data that are transferred through the connector can be enriched by additional attributes (e.g., time stamps and state information) or reduced by applying predefined rules to decrease the number of attributes of the original data. Another operation that changes the data during transportation is content translation. Spitznagel states: “When two components agree on the semantics of a communication protocol but disagree on details such as the format or unit of the data to be communicated, a data translation may be used to resolve the mismatch” [36].

For the representation transformation of data, the two disjoint facets permanent and temporary transformation are relevant.

A language-to-language translation takes place when data that are described in one language must be transformed into another language without altering content. If the character set in which the data are encoded is changed while it is transferred through the connector, an encoding-to-encoding translation occurs. Both are permanent transformations because the change in data representation extends beyond the transportation of data through the connector.

If the data rather than the connector are secured, an encryption of the content is required. After entering the connector, the data are translated into a secured intermediate format and will be decrypted before reaching the target endpoint [24]. An identical temporary transformation occurs when the data are compressed at one endpoint of the connector and will be uncompressed before leaving at the other endpoint [11], [25], [2].

4. BENEFITS OF THE SPECIFICATION SCHEMA
Using the specification schema for software connectors is helpful for several purposes that range from evaluation of modelling to documentation. In particular, the schema offers advantages in performing the following tasks:

- comparing connectors (evaluation)
- choosing connectors (selection)
- designing connectors (modelling)
- describing connectors (documentation)
- creating connectors (synthesising)
- establishing connectors (generative)

For a set of given connectors, it is frequently difficult to decide which connector best fulfils the requirements and is most appropriate for a particular envisioned architecture. Thus, the specification schema helps compare connectors according to their attributes and leads to the selection of the connector that fits best.

The specification schema is reasonable not only when connectors exist but also under circumstances when connectors have yet to be developed. At least four situations can be addressed and supported by the schema. First, for an architect, the descriptive structure of the schema can act as a checklist to reconcile system requirements with connector attributes at different levels. The different views and levels of the specification schema ensure that the relevant connector properties are not omitted. Second, the specification schema supports the documentation of connector attributes after design decisions have been made. The specification schema forms a standardised documentation template for fixing connector features during or after design and development and presents a clear and unified structure for discovering the characteristics of existing connectors in the case of connector adaptation. Third, by combining reasonable attributes of different dimensions, new connector types can be synthesised that represent previously unknown connector aspects [29]. Fourth, because the specification schema describes “what” are the attributes of a connector and not “how” a connector is implemented in a certain technology, this high-level description can act as the starting point for a generative approach. The schema can be related to model-driven architecture [31] such that it represents a platform-independent model (PIM) for connector generation. Based on this PIM, a translator for a targeting technology can be described that generates concrete connectors for a specific runtime platform, thus realising a platform-specific model.

5. EXAMPLE SPECIFICATION OF A CONNECTOR

To illustrate the use and feasibility of the proposed specification schema, the schema will be applied to the design of a connector between Twitter and a fictitious application.

Today, micro-blogs of companies and private users are a widely used type of service to stay informed of events in both business and private areas. One big player in the micro-blogging domain is Twitter [38], known for its simple use of spreading short text information (aka tweets) to interested users, called followers. To provide its services, Twitter offers an application programming interface (API) that covers four main use cases [16]. First, a user can send a tweet to its micro-blog to update its current status. Second, interested followers who have subscribed to the blog can retrieve tweets when they are online. Third, users can search for tweets using predefined topics called hashtags. Fourth, before using these services, the user must authenticate itself.

A software connector that coordinates and communicates between the Twitter API and an application must integrate these four services. A design that incorporates all four use cases can either interconnect the application and Twitter by establishing a separate connector for each service or by integrating them into a complex connector. For the following discussion, a complex connector is assumed that is separately developed for an individual application and is not provided by Twitter as a standard connector but can be instantiated for various applications.

Figure 3 depicts the design of this connector and shows the services that the connector mediates between Twitter and the application and the internal structuring of this connector.

For reasons of clarity, only the attributes of the specification schema and their selected values are listed in Tables 1 and 2, and a detailed discussion of other aspects of the schema is left to the subsequent textual description. For the same reason, only the use cases “send tweet” and “receive tweets” are selected to apply the specification schema to the Twitter connector.

5.1 External view
From the outside, the attributes of the connector that are required by the interconnecting components are in view. The space and time considerations of the connector must be considered to fulfil the given requirements. The space dimension of the external view can be described by the attributes location, cardinality and directionality. Because Twitter and the fictitious application are distributed in different runtime environments, a remote connector is required. Because the connector is separately designed for an interconnection between an individual application and Twitter, a one-to-one relationship between them can be assumed. This connection thus forms a unidirectional communication that is initiated by the application.

How the communication takes place, what type of processing mode is required and how durable the connection should be are discussed in the time dimension of the external view. Different communication types are required for selected use cases. To add a tweet to the micro-blog, a discrete message of limited length is asynchronously sent to Twitter, whereas to retrieve tweets, a continuous connection in which several tweets are sent by Twitter to the application must be opened. For the user of the application, it is faster if messages can be sent while new tweets are retrieved at the same time (parallel processing). Maintaining a durable connection between Twitter and the application enables the user to use the services provided one after another without creating a connection every time a service is invoked.
Table 1. External view of Twitter connector

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Send tweet</th>
<th>Value</th>
<th>Receive tweets</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>remote</td>
<td>remote</td>
<td></td>
</tr>
<tr>
<td>cardinality</td>
<td>1:1</td>
<td>1:1</td>
<td></td>
</tr>
<tr>
<td>directionality</td>
<td>uni</td>
<td>uni</td>
<td></td>
</tr>
<tr>
<td>communication</td>
<td>discrete (asynchronous)</td>
<td>continuous (isochronous)</td>
<td></td>
</tr>
<tr>
<td>processing mode</td>
<td>parallel</td>
<td>parallel</td>
<td></td>
</tr>
<tr>
<td>durability</td>
<td>durable</td>
<td>durable</td>
<td></td>
</tr>
</tbody>
</table>

5.2 Internal view

The attributes of the connector that are essentially independent of the components are the focus of the internal view. Because of the many attributes that can be assigned to internal connector properties, the internal view consists of more levels. To group the view at a high level, the dimensions of structure, behaviour (control flow) and information (data flow) have been introduced.

The structure dimension indicates how a connector is set up (horizontal criterion) and whether technically lower-level services are used (vertical criterion). The structural organisation of the Twitter connector is depicted in Figure 3. This complex connector consists of four simpler connectors, each mediating a service between Twitter and the application. These functional units are horizontally arranged between the endpoints of the connector. Instead of creating these interconnection services from scratch, platform services (e.g., the Spring framework and its social media subproject) at technically lower levels may reasonably be utilised.

The behaviour dimension describes how the control flow is managed between the components. The endpoint-oriented criterion groups attributes that can be assigned to both ends of the connector. Here, the interaction, content transfer strategy invocation mode and protocol adaptation are important to specify. Because there is no superordinate component that integrates Twitter and the application as a starting point for coordination in the exogenous connector, an endogenous connector is assumed to be invoked by the application. To add a message to the micro-blog at Twitter, a “push” content transfer strategy is used at the endpoint towards Twitter, initiating the transmission of a short message. The opposite is true when tweets from Twitter are received using a “pull” transfer strategy. It can be assumed that the Twitter API provides an explicit method invocation mode for sending tweets to Twitter, whereas spreading tweets to followers is reasonably accomplished by using an implicit mechanism such as event-delegation. Coordinating components is forced by protocol considerations. At the endpoint towards Twitter, the connector must use an advanced protocol over http (e.g., WebService), whereas the application expects to use a protocol incorporated in the programming language (e.g., Java). Therefore, the Twitter connector must manage protocol-to-protocol conversion for all four use cases. The data-oriented criterion focuses on the control flow within the connector that is neither influenced by the endpoint of the connector nor the information to be transmitted. Attributes such as transaction, security, loading strategy, the type of data transfer path and filtering are defined. There is no need for a transactional mechanism inside the connector because every action can be considered atomic in nature. Tweets, which go to and from Twitter, are generally for public use, but a secured connection from the application to Twitter is reasonable to guarantee privacy concerns. Different design decisions are necessary for sending and retrieving tweets, depending on their delivery. To send a tweet, no buffering is required, and an exception is raised if a message cannot be added. Paging is mandatory for fetching tweets from Twitter. The number of tweets assigned to a hashtag and later retrieved from Twitter can be enormous and can only be reasonably handled by a paging mechanism, but no such mechanism is required when sending messages to Twitter. Adding a message to the micro-blog at Twitter via the connector normally requires no filter to check messages against predefined criteria. For retrieving tweets from Twitter, such a filtering mechanism sometimes makes sense if a follower has subscribed to a large number of micro-blogs, and a filter using certain keywords might eliminate uninteresting tweets.

The information dimension comprises the criteria “data” and “function” and answers the questions of how information is made up and what functions can be applied to information. To differentiate the data criterion, attributes such as composition, payload type and volume are important. Twitter allows the sending and retrieving not only of short text messages but also of photos. If both media types are included in a message, the tweet represents “complex” data that the connector transmits. In such a complex message, there are two different types of payload. The plain text of the message belongs to the “text” payload type category, whereas a photo represents “binary” data. It can be assumed that the data volume to be transferred to Twitter is at a lower level, whereas receiving tweets from Twitter is in the range of low to mid-volume. The type of functions that operate on the data interchanged by the connector is the focus of the function criterion and is detailed by the preparation and transformation attributes. Data can be manipulated by division, enlargement or translation, as indicated by the preparation attribute. The three sub-attributes of the preparation attribute are not applied to the Twitter connector because it is assumed that the application sends and receives messages without manipulation. No division of a tweet makes sense; no extra values have to be added to the tweet, and the same defined content format for text messages is processed in both applications. The last function to be considered is content transformation. The permanent sub-attribute indicates that the change in data representation lasts longer than the transportation of data through the connector. For the Twitter connector, it can be assumed that a tweet, represented in plain text like XML or JSON, must be transformed to a Java object for the fictitious application. However, no encoding is required because the content of the tweets can be supposed to be in a frequently used type of code (e.g., UTF-8). Temporary transformations are functions such as encryption and compression, which translate the data in an intermediate format while it is transported through the connector. The tweets need not be encrypted because the Twitter connector is secured. However, because tweets can contain not only plain text but also photos, they will be compressed inside the connector after they are received from Twitter and will be uncompressed before delivery to the application.

Table 2. Internal view of Twitter connector

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Send tweet</th>
<th>Value</th>
<th>Receive tweets</th>
</tr>
</thead>
<tbody>
<tr>
<td>connector set-up</td>
<td>composed</td>
<td>composed</td>
<td></td>
</tr>
<tr>
<td>layering</td>
<td>layered</td>
<td>layered</td>
<td></td>
</tr>
<tr>
<td>interaction</td>
<td>endogenous</td>
<td>endogenous</td>
<td></td>
</tr>
</tbody>
</table>
6. CONCLUSION AND FUTURE WORK

This paper has presented a hierarchical specification schema for software connectors. The major difference between this schema and other proposals is that it differentiates between external and internal views of connectors and guides the architect so that the specification process of a connector begins at a conceptual level. The external view allows a rather coarse-grained description of a connector, focusing on attributes required by the interconnecting components. Detailed characterisations of attributes that are realised inside the connector are mostly independent of component constraints and are described by the internal view. Both views of the presented schema begin at a conceptual level and proceed through a stepwise refinement of characterising connector attributes that are described independently of technical considerations, until concrete attribute values are reached. The benefits resulting from the new specification schema have been identified and described. Finally, the schema was successfully applied to the design of a Twitter connector, which interconnects a fictitious application and Twitter.

The intended focus of the proposed specification schema is to describe connectors at the conceptual level, ignoring technical considerations. Nonetheless, within a software project, concrete connectors must be derived from architectural design to be realised in a given software environment. How the specification schema might be used in conjunction with a collection of existing connectors in a concrete software platform (e.g., the JEE platform specification) is a question for a future research paper.

7. ACKNOWLEDGMENTS

I would like to thank Stefan Kleine Stegemann for his scrutiny of the approach and for proposing Twitter as an example for applying the specification schema.

8. REFERENCES


