异构系统开放封装的技术分析与实现框架

张晓林、李广建、曾蕾、毛军、刘炜

文摘：开放封装是一种将异构系统封装为可以标准表示、开放识别和公共利用的开放系统的机制。本文对开放封装的意义与性质、开放封装的层次、以及包括元数据转换、元数据封装、公共检索协议、开放系统界面等在内的开放封装技术进行了初步探索。

关键词：异构系统、开放封装、元数据转换、元数据封装、检索协议

Analysis and Framework of Open Wrapping for Heterogeneous Systems

Abstract: Open wrapping is a mechanism to wrap up heterogeneous systems into open ones through standard expression, open representation, and public registration. This paper analyses the nature and use of open wrapping, the layers of wrapping required, and the techniques used for open wrapping such as metadata crosswalks, metadata wrapping, open search protocols, open interface languages.

Keywords: Heterogeneous systems, Open wrapping, Metadata Crosswalks, Metadata Wrapping, Search Protocols

1．开放封装的意义

现实系统环境是一个由众多分布、异构和自主的资源系统组成的开放环境，现实应用则要求对这些系统进行整合来向用户提供逻辑化甚至个性化的集成服务。

为了形成有效的集成服务，我们可以要求新建设的资源系统采用标准的元数据格式、检索方式、检索记录调用机制、检索记录传输格式和传输协议，从而支持整合检索、开放链接和基于 WEB 的集成使用[1]。但对于已经存在的资源系统和第三方资源系统（在科学院情况下这些系统可能占多数），它们往往各自具有自己的数据结构、检索方式、检索与调用协议、数据记录传输格式等。我们不可能要求这些系统都按照我们规定或选择的方式组织它们的系统，只能将它们的异构性用某种方式予以屏蔽或转换，使它们“ 看起来 ”象同构系统（或标准系统），从而支持集成操作。

要达到这点，还需要解决另一个概念问题：谁是异构？严格来说，一个系统在某个方面的特殊性对于在这个方面采取其他方式的系统而言就是异构系统，因此异构实际上是相对的，或者说任何系统都是异构系统。那么，我们以谁为基准来决定“ 标准 ”系统？或者说我们要按照哪个系统的标准来屏蔽或转换其他与它不同的系统？长期以来，图书馆界倾向于用自己系统作为标准，例如有些数字图书馆体系以 MARC/Z39.50（M/Z 系统）[2-3]作为标准，为其他非 M/Z 系统建立转换界面，使其他系统看起来像一个 M/Z 系统。但这存在以下问题：

（1）从 Internet 环境看，M/Z 系统远不是主流机制；进一步，M/Z 系统是以图书馆书目数据系统为中心的机制，应用到以数字内容对象和以 Web 机制为基础的数字图书馆环境中是否有效，显然存在很大问题。

（3）更进一步，数字图书馆环境中的集成操作可能从不同的用户应用系统角度进行，例如从用户课题网站、LMS、CSCW、WFML、或者用户所在机构的数字图书馆系统或个人的数字图书馆系统。从任何一个应用系统角度讲，开放封装就是将其它不同的系统封装成与本系统一样的同构系统，使本系统能够对它们进行集成操作。我们当然可以针对某一个用户应用系统的某一个集成操作，按照该“ 本系统 ”方式建立相应的封装转换机制，将其它系统转换成本系统的“ 同构系统 ”。但由于应用系统的多样性、异构系统的多样性、集成操作的多样性和集成操作的交叉性，这种个别转换方式实现成本高、可伸缩性低、可扩展性低。

因此，我们必须寻求一种公共方法，在不改变异构系统内在方式的情况下，将异构机制

封装为一种可以标准表示、开放识别和公共利用的逻辑系统，使得任何能够采用同样的标准表示和开放识别方式的用户应用系统，可以把任意异构系统看成是同构系统，通过同样的标准方法去发现、检索和调用其资源和服务。这就是开放封装的意义和目标。

2．开放封装的层次和要求

开放封装要将系统的异构方式转变为“标准表示、开放识别和公共利用”的方式，那么一个系统有哪些可能的异构方式？当然，系统操作系统、应用软件语言、数据库系统等都可能“异构”，但它们更多的是内部机制。对于用户应用系统而言，关心的是系统交互中体现的异构性，只要在交互界面能消除“异构”，至于各个系统内部如何就可以不必考虑[4]。因此一个资源系统的异构性可能表现在：

（1）数据内容对象结构，例如编码格式、文件格式、对象组织结构（例如由多个多媒体文件组成的课件）、标识方式等。

（2）内容对象的元数据定义（这些元数据本身也是内容对象）。

（3）内容对象的检索界面，主要涉及检索的逻辑表达方式，例如检索点（类别、属性、命名）、检索式组织与表示方式等，还涉及检索结果的处理、组织和显示方式。

（4）内容对象的检索协议，包括检索式和检索内容封装方式、检索内容调用方式、检索内容传输方式。

系统的异构性可能表现在上述任何一个层次，也可能几个层次同时存在，例如一个系统采用 XML[5]表示 DC[6]元数据记录、采用 LDAP[7]协议进行检索，而另一系统采用 M/Z 方式，两者的异构性至少表现在元数据定义、元数据记录封装、检索协议等方面。因此，开放封装以及转换操作可能针对不同的层次或层次组合（这也意味着开放封装本身需要模块化设计）。

开放封装要将系统的异构方式转变为“标准表示、开放识别和公共利用”的方式，接着需要解决的就是：什么是标准表示、开放识别和公共利用？

（1）“标准表示”主要指我们能把各个系统上述层次的异构表示转换成一个能够公共理解的共同逻辑内容和公共表现形式。由于异构性涉及逻辑（例如 DC 与 MARC 关于内容的定义与划分）和形式（例如 XML 封装和 ANS.1 封装[8]），所以“标准表示”中一是要解决逻辑上的异构性，一是要解决形式上的异构性。因此，开放封装也可能涉及两个方面。某个开放封装模块可能只解决逻辑异构性或形式异构性，也可以同时解决两者。无论怎么解决，所选择的“标准表示”方式都应该根据主流环境、发展趋势和成熟程度来确定。

（2）“开放识别”是指封装结果能够被普遍地和自动地识别，支持用户应用系统方便地利用封装机制来与异构系统互操作。其实，这与形式异构性方面的“标准表示”非常相关，目前网络环境似乎已经给出一个比较明确的选择，即基于 XML 的机制。当然，采用 XML 语言并不意味着我们必须采用 XML 方式去统一地描述每一个转换封装细节，而是遵循 XML 机制所规定的基本原则、标记方法、语义表示方法、扩展方式、转换机制等。不同系统可以在此基础上采用符合自己需要的封装方式来处理逻辑异构性和形式异构性，但由于 XML 语言的开放性和可解析性，各应用系统可以对用 XML 描述的封装信息进行识别和解析。

（3）“公共利用”要求开放封装所形成的封装模块可公共获得和可开放解析。可开放解析依赖“开放识别”，而可公共获得则要求封装模块遵循开放描述的原则，或者被置于被封装系统的公知位置供第三方系统进行搜索，或者到公共登记系统[9]进行登记以支持公共调用。当应用系统发现一个异构的资源系统后，可以查询该异构系统的开放封装模块，调用这些模块嵌入到自己的系统流程中，通过这些模块与该异构系统进行互操作。实际上，当一个系统在公共服务登记系统上登记自己系统或资源时，应该同时登记自己的开放封装模块；如果开放封装模块由第三方开发，则该第三方应该向公共登记系统登记这些模块。

3．开放封装的可能方式

根据异构性层次，开放封装可能涉及数据内容、元数据、检索协议、传输协议等。我们首先讨论对不同层次异构性进行封装的具体方式，然后讨论结合几个层次的封装方式。

（1）关于数据内容的开放封装，主要涉及对内容对象的编码方法、文件格式和标识方式进行标记，以便作为第三方的应用系统知道该如何处理它。所谓标记，是在系统信息或内容对象中关于编码、格式和标识符等的明确信息，例如 MARC 头标区关于编码标准的说明、XML 文件的编码语言属性、HTTP 信息中关于文件格式的说明语句等。标记信息作为开放封装工具，要求本身具备可获得性和可解析性。如果这些标记信息本身是采用专用方法标记的，应该采用标准表示方法来揭示这个信息，使得第三方系统能够方便地释读这个信息。例如，建立开放描述的元数据来记载关于内容的标记信息（例如 DC 中记载 FORMAT），在内容对象上附加开放描述的标记信息（例如 GEDI 标准[10]专门设立一个头标字段，用开放语言标记所传输的数字文件的编码语言、文件格式）。而且，如果文件格式是特殊的（例如 GIS 文件、多媒体课件、复杂数字对象等），应该采用标准方法标记和指向能够释读这个格式的专门模块，使应用系统能够调用这个模块、将其作为插件嵌入应用系统、从而处理这个特殊的文件。对于逻辑形式或本地方式的标识符，也应指向对应的标识符解析系统。

如果数据内容是复杂数据对象，往往需要设计相应的开放组合和封装机制，典型的是 ADL/SCORM 的 Content Aggregation Model[11]和 LC 的 Metadata Encoding and Transmission Standard[12]。它们都利用 XML 语言描述了复杂对象的内容成分、组合结构和相应元数据对象，支持 XML 的应用系统可以方便地识别和解析由此封装的内容对象。

（2）关于元数据内容异构性的开放封装，可以采取的方法可能包括：

A．将异构系统的元数据格式转换为某种“ 标准 ”格式（Metadata Crosswalks 或 Metadata Mapping），例如 NSDL 要求所有参与资源系统（无论其本身元数据格式是什么）都要提供 DC 格式的元数据记录[13]。被采用的“ 标准 ”格式应该在元素组成、元素语义定义和应用领域等方面有足够的普遍适应性，本身有较广泛的应用，多数系统能够识别。当然，这种作为“ 中介 ”性质的标准格式可能与一般意义上的标准格式不一样。由于异构元数据转换中存在语义损失，越是详细、精细的元数据格式越容易在转换中产生语义损失，越难承担“ 公共中介“ 的任务。理想的“ 中介 ”格式能够为各个异构元数据提供一个容易比照、语义转换比较容易的公共核心集。“ 中介 ”格式主要支持集成操作，而集成操作主要是为资源发现、检索和调用服务，不必也难以承担更为复杂的资源编目、管理和长期保存等这些本来就是各本地异构系统的自主任务。为了支持这种转换，资源系统应该建立本系统元数据格式与“ 标准 ”格式的标准转换关系和实现模块，在自己的元数据记录和元数据管理系统中予以指向。当然，资源系统也可以寻找、验证和链接第三方元数据转换模块，支持本系统元数据格式与其他流行元数据格式的转换，并予以有效指向。

B．将异构系统的元数据元素与某个公共的元数据字典连接起来，通过该字典准确定义元素语义和元素关系，从而支持那些理解该字典的系统了解这些元素语义和元素关系，在此基础上将本地元数据转换为其他元数据格式。INDECS 系统的元数据字典[14]、NISO Metadata for Image[15]数据字典、ebXML 的 Core Component Dictionary[16]等就是这类情况。

C．更为理想的情况是元数据本身指向自己的定义链，包括元素语义定义、语义定义所依据的语义体系、语义体系所依据的概念体系、概念体系所依据的上层概念体系。至少，应该基于 XML Schema[17]定义元数据格式，基于 RDFS[18]定义元数据元素语义关系。这样的元数据在定义层面就是开放的，支持第三方对自己定义链的解析。

（3）关于元数据的形式封装，主要是指将用本地形式表示的元数据格式（尤其是其结构表示和语法表示）通过一个可以开放识别的标记（封装）方式表示。通常，资源系统通过自己的专用格式和方法向自己客户端提供数据，或者以专有模板来组织元数据提供给浏览器

用户。前种方法的不开放性是显而易见的，即使是后一种方法，也往往不能准确地识别元数据的完整结构和语义（因为提供的往往是孤立的元素或元素组）。

开放封装要求封装形式要求让应用系统能读懂元数据的格式（结构与语法），趋势是将 XML 作为公共语言来定义和描述元数据，将内部格式元数据转换为用 XML 语言描述的元数据。这又有两种可能，完整对应转换或提取部分内容重组转换。

MARCXML[19]即是将 MARC 记录完整地用 XML 语言封装，从而支持 xml-capable 系统来解读 MARC 记录，也支持基于 XSLT[20]的转换系统将 MARC 记录转换为其它元数据格式。

MODS[21]则提取 MARC 记录中的部分内容，按照 XML 语言方式定义为一个新的元数据对象，其中的元素可能进行了重组。

其实，现在许多元数据格式尽管在其系统内部可能采取专门的数据形式（例如数据库格式），都定义了用 XML/DTD、XML Schema、RDF 等进行描述的方法，并以这些方法构造向外界提供的数据。例如，OAI[22]机制规定所有支持 OAI 协议的资源系统将内部元数据转换为 DC 格式（逻辑封装），并且用 XML 形式标记元数据内容（形式封装）。

我们希望，资源系统在专有客户端/服务器机制外，提供一个基于 XML 的支持浏览器的检索界面，提供用 XML 表示的元数据格式（以及对这样的元数据定义文件的链接）。

（4）关于检索界面的封装。集成操作的主要用途是发现、检索和调用，因此开放封装的重点也在检索界面。检索界面所涉及的内容包括检索点（类别、属性、命名）、检索式组织与表示方式、检索结果表示方式、检索结果调用方式等。

Z39.50 作为图书馆界普遍应用的异构系统检索协议，实际上是一种检索转换机制，将异构系统的检索指令和结果通过一个公共的方式来表达，从而支持异构系统间的检索操作。在 Z39.50 过程中，检索双方需要选择共同的检索参数集（Bibliographic Attributes Profile）、检索式形式（Query Type）和记录形式（Preferred Syntax）以及一些技术细节，以此来保证互操作。由于 Z39.50 是面向复杂检索、主要支持 M2M 操作、依赖特定的客户端/服务器以及它们与受主系统的绑定，一般需要一对一的事先设定，而且本身并不能低成本地被识别和解析，实际上是图书馆领域的专有检索协议，其面向连接、通过多个不同的 APDU 分别承担不同功能操作、通过 ASN.1/BER 方式封装 APDU 数据等，使其在通用领域、一般检索和 P2P 检索中的应用受到很大局限，难以成为一个公共的转换机制。也正因为这个原因，ZIG 才开始探索适应开放环境的 Z39.50 检索技术，包括基于公共检索语言（以及相应的检索式方式）、采用 SOAP[23]封装检索式，依赖于 HTTP 传输协议，另外还有人提出用 XML 语言封装 Z39.50 编码方式（XER）[24]。

SDLIP[25]采用界面定义语言（Interface Definition Language）来定义检索系统的三个界面：Search Interface，Result Access Interface，Source Metadata Interface。在检索界面里，SDLIP 通过 XML 对象来描述检索子集、检索式、检索结果属性集、检索结果等，并定义了若干控制参数；检索式和检索结果属性集的语义可通过外部命名域（Namespace）定义，也可在 XML 对象内部定义。用 IDL 描述的界面可以被映射到 DASL 和 CORBA 协议中。SDLIP 还可通过 Source Metadata Interface 来查询资源系统的子集集合和各个子集的属性集合，这些集合的描述信息都是 XML 对象。

DASL（Distributed Authoring, Searching, and Locating）[26]是 WebDAV 的一个扩展部分，试图建立一个基于 WEB 的支持数据库检索（而非网页检索）的公共检索协议。

特别值得注意的是 WSDL[27]，它基于 XML 来定义一个系统可以支持的远程程序调用（Remote Procedure Call）。首先，WSDL 定义一组数据元素（types/elements），这些数据元素可以表达检索指令（例如属性-值对）；WSDL 将这些数据元素组织到消息（message）中，作为系统逻辑操作（portType）的输入或输出参数（input/output）；WSDL 进一步将这些操作及其相应消息被绑定在一定的传输协议（protocol binding）上，并通过传输协议在指定的

端口（port）接受，而一个服务系统（service）可能包含多个端口以及相应的逻辑操作。因此，一个系统可以建立用 WSDL 语言表示的检索界面描述。至于检索指令的语义可以通过外部命名域或内部详细定义的 types/elements 等细化。

DASL 和 WSDL 方法所建立起来的检索界面的开放性应该优于 Z39.50 和 SDLIP，尤其在检索指令和检索对象元数据已被明确定义和指向时。因此，当一个系统利用这些方法建立检索界面的开放封装时，需要建立明确的关于检索界面、检索式和检索对象元数据的定义与描述，这些描述信息应该基于 XML，应该被置于本系统的公知位置供开放搜寻，或者向公共登记系统登记以支持公共检索。第三方系统可以通过这些描述来解析检索界面、检索式结构、检索对象元数据结构，从而配置自己的检索系统（甚至可以自动配置）。

参考文献：

[1] 张晓林。数字图书馆建设中的开放描述机制。现代图书情报技术, 2002.3
[2] MARC Standards http://www.loc.gov/marc/
[3] Information Retrieval (Z39.50-1995): Application Service Definition and Protocol Specification http://lcweb.loc.gov/z3950/agency/markup/markup.html
[4] e-Government Interoperability Framework (e-GIF). http://xml.coverpages.org/egif-UK.html
[5] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000. http://www.w3.org/TR/REC-xml
[6] Dublin Core Metadata Element Set, Version 1.1: Reference Description, July 2, 1999 http://dublincore.org/documents/dces/
[7] Wahl, W., et al. Lightweight Directory Access Protocol (v3), RFC2251, December 1997 http://www.ietf.org/rfc/rfc2251.txt
[8] ASN.1 Homepage. http://www-sop.inria.fr/rodeo/personnel/hoschka/asn1.html
[9] Heery, R. and Wagner, H. A Metadata Registry for the Semantic Web, D-Lib Magazine, v8, n5, 2002 http://www.dlib.org/dlib/may02/wagner/05wagner.html
[10] General Electronic Document Interchange. ISO/DIS 17933, Dec. 15, 1998. http://www.rlg.org/gedistand99.pdf
[11] Advanced Distributed Learning. Sharable Content Object Reference Model. Version 1.2. Oct. 1, 2001. http://www.adlnet.org/ADLDOCS/Document/SCORM_1.2_CAM.pdf
[12] Metadata Encoding and Transmission Standard. http://www.loc.gov/standards/mets/
[13] NSDL Metadata Primer. http://metamanagement.comm.nsdlib.org/outline.html
[14] INDECS Metadata Framework, June 2000. http://www.indecs.org/pdf/framework.pdf
[15] NISO Z39.87-2002. Draft Standard for Trial Use. Data Dictionary - Technical Metadata for Digital Still Images. June 1, 2002. http://www.niso.org/standards/resources/Z39_87_trial_use.pdf
[16] ebXML Core Component Dictionary. May 10, 2001. http://www.ebxml.org/specs/ccDICT.pdf
[17] XML Schema. http://www.w3.org/XML/Schema
[18] RDF Schema. W3C Working Draft 30 April 2002. http://www.w3.org/TR/rdf-schema/
[19] MARC XML. http://www.logos.com/marc/marcxml.asp
[20] XSL Transformations (XSLT). V1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xslt
[21] MODS: Metadata Object Description Schema http://www.loc.gov/standards/mods/
[22] Open Archives Initiative. http://www.openarchives.org/
[23] Simple Object Access Protocol (SOAP) 1.1 W3C Note 08 May 2000. http://www.w3.org/TR/SOAP/
[24] XER (XML Encoding Rules). http://asf.gils.net/xer/
[25] Simple Digital Library Interoperability Protocol. http://www-diglib.stanford.edu/~testbed/doc2/SDLIP/
[26] World Wide Web Distributed Authoring and Versioning. http://www.ics.uci.edu/~ejw/authoring/
[27] Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001. http://www.w3.org/TR/wsdl

本文最初发表在《情报理论与实践》2003 年第 3 期第 260-263 页。
本文作者为张晓林、李广建、曾蕾、毛军、刘炜。