# A Service-Oriented Approach for Integrating Broadcast Facilities

Alexis Huf, Ivan Salvadori, Frank Siqueira
*Graduate Program in Computer Science, Department of Informatics and Statistics*
*Federal University of Santa Catarina - Florianópolis, Brazil*
*alexis.huf@posgrad.ufsc.br, ivan.salvadori@posgrad.ufsc.br, frank.siqueira@ufsc.br*

***Abstract*—Television broadcast production facilities capture, manage, edit, handle, and broadcast audiovisual content by using a wide array of specialized equipment and software. The complex workflow in this environment demands interoperability between devices, but vendor-neutral protocols do not provide access to a significant amount of functionality. This paper proposes the adoption of a Service-Oriented Architecture for controlling broadcasting equipment, addressing difficulties specific to this environment such as the prevalence of non-Web Services, embedded devices, and constrained computational resources. The proposed solution is centered on a semantic service registry, which is able to compose mediators and produces stubs in response to service selection requests. The prototype registry is experimentally evaluated in simulated scenarios, focusing on how size and complexity of the broadcast facility impact on response times.**

***Keywords*-service composition; broadcast automation; system integration; OWL-S.**

## I. Introduction

A television broadcast facility [1], [2] has as its main purpose to continuously broadcast audiovisual content using ground based antennas, satellites or cables. Many of these facilities not only retransmit signals from a television network, but also include locally produced content, such as news programs or advertisements. To produce the television broadcast as scheduled, these facilities employ several types of specific-purpose devices and software that capture, edit, store and manage both live video signals and video material.

As an example, consider the production of a live news program. The anchor will usually introduce a particular news item before the broadcast video signal changes from the studio to a previously produced report. In more detail, the anchor will be reading the story text from a teleprompter, which may have been recently edited by a journalist in another room. The report, most likely a file on a storage server, must be transferred to a playout device to transform it into a video signal. The switch from the anchor's video signal to the report video signal is done by the master switcher, which may also apply a transition effect and perform adjustments on audio and video.

These processes, or workflows, in broadcasting jargon, are ultimately specific to each broadcaster and can be rather complex. Usually workflows combine human interaction, device-to-device control, and automation software, the last two respectively resembling static, vendor-defined choreography and orchestration. Since a large portion of the devices offer only proprietary control protocols, the workflow itself may become more complex, as well as its formal definition.

This heterogeneity in device control protocols, embedded software, and the overall constrained computational resources, are challenges to the direct application of most SOA technologies. With these challenges in mind, we propose a service repository capable of selecting and composing services. An ontology based on OWL-S [3] is proposed to model concepts and elements that deal with heterogeneity. A distinct feature is that not only Web Services are allowed, but also services accessible through proprietary protocols or serial communication lines. To allow for heterogeneity in data representations, in protocols, and in the offered services, the registry composes data and service mediators into stubs, which are handed as replies to service queries.

This paper proposes the adoption of a Service Oriented Architecture, centered on a service repository, for the television broadcast environment. The applicability of the proposed architecture is evaluated with respect to how size and complexity of a broadcasting facility impact the time required by the repository to compose stubs. The proposed solution aims to tackle challenges - such as heterogeneity, legacy protocols, and constrained resources - present in the broadcasting scenario, as well as in small office and home automation.

The remainder of this paper is structured as follows. In section II, a brief description of broadcasting automation and related work from the broadcasting field is presented. Challenges specific to this environment are described in section III. Related work from the SOA literature is presented in section IV, and the proposed integration architecture is described in section V. Experiments are discussed in section VI, followed by the concluding remarks in section VII.

## II. Broadcasting Automation and Protocols

Automation in broadcasting environments is not a novel concept. Early systems, like the one described in [4] focused on automating sequences of video signal switching and equipment triggers. Under these systems, equipment control was performed by electrical pulses and integration consisted of wiring the devices and occasional adapter circuits. The control technology evolved from wiring to protocols over

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/scc.2016.97

serial lines (such as the ones described in [5]), and later, to protocols over IP networks. However, control over serial ports and electrical pulses, popularly known as GPI (General Purpose Input), are still used on most new equipment.

Most of the protocols used in broadcasting production were designed by a single vendor for use in its own products. As a requirement for integrating equipment from two vendors, one of the vendors has to implement the protocol of its counterpart. Since this is a costly approach, when the first NCS (Newsroom Computer System) were about to become largely available, the Associated Press (developer of the ENPS NCS) proposed the development of the MOS (Media Object Server) protocol [6] as an open protocol, with participation of all major vendors. The MOS protocol is based on XML messages transmitted directly over TCP connections, which allow the NCS to query, cue and play media in media object servers and allow media object servers to interact with the news rundown in the NCS. MOS is the most widely deployed vendor-neutral broadcasting protocol, but it covers a limited scope and some devices do not have the resources to directly implement it, as exposed in section III. In addition, NCS vendors often offer proprietary protocols offering functionality not present in MOS.

Interoperability is a strong requirement by broadcasters, and several standards were produced to this end. MXF (Media eXchange Format) [7] is a media container developed for broadcasting applications that is widely adopted and supported. Similarly, the development of technologies used for video signal distribution in studios was largely driven by standardization. For broadcast automation, vendor-specific protocols have always been the norm. However, the Society of Motion Picture and Television Engineers (SMPTE) has recently published two standards that relate to automation and device control: BXF (Broacast eXchange Format) [8], and MDCF (Media Device Control Framework) [9]. BXF was designed for the exchange of playout schedules, in addition to extensive metadata. This protocol allows the production of as-run information, used for billing advertisers, and the request by devices for the provisioning of content on the schedule (the actual provisioning protocol is not specified). MDCF, on the other hand, has a larger scope, providing identification, description, and discovery for media and media devices. Under MDCF, devices expose a dynamic set of capabilities, which can be invoked as Web Services. The standard defines capabilities that handle description, security and search functionality, but it yet does not define any actual capability that pertains to media devices.

Historically, broadcasting automation has been focused on five not strictly disjoint subareas: break and playout automation [4], [8], [10], [11], news production [6], media asset management [12], multichannel operation [13] and master control [2]. For many devices, such as cameras [14] and video routers [2], it is common that instead of being remotely controlled by an automation system, panels produced by the same vendor are used by operators to this end. For automation systems, device control is only one of the concerns of the controller device, the other being managing the broadcaster's data, such as schedules, news items, or media rights and meta-data. In these subareas, the common approach is for the controlling device to enact processes statically defined by their designers using the data that it manages. These static processes, albeit reconfigured, are prone to become misaligned with changes in the broadcaster's workflow, requiring, at best, that parts of the workflow be performed by humans.

In broadcasting environments, devices that offer a control protocol can be seen as offering a service in a more liberal sense. Since there is no common protocol and services are not self-described, a service-oriented environment for broadcasting, including process management, is still not a common reality. An important step in this direction, together with MDCF, is FIMS (Framework for Interoperable Media Services)[1] [15], which defines a set of Web Services to support media production, covering capture, transference, transformation, quality assurance and media repositories. FIMS is a clear departure from the traditional black box automation system, given that the specification clearly intends the services to be controlled by an orchestration system.

## III. Challenges for SOA adoption in broadcasting environments

Unlike other businesses, such as retail or consumer services, SOA had a slow adoption in broadcasting. While Web Services are present, they are far outnumbered by proprietary protocols, some of which use serial communication lines, limiting the communication to only the two participants wired together. In the realm of IP, a problem is that while MOS, the most popular protocol, specifies a parallel version [16] based on SOAP Web Services, only one of the two major NCS vendors supports it. That vendor lists [17] only 2 devices compatible through the SOAP variant, against 62 devices compatible through the sockets variant of MOS.

Many of the devices used in broadcasting also have no additional resources to host Web Services; this is the case for almost all devices that output or receive video signals. Embedded devices offload real-time video processing to specialized integrated circuits and FPGAs (Field Gate Programmable Arrays), but their microcontrollers do not have enough resources to host a typical application server. Devices based on server-grade PC hardware, on the other hand, often demand large amounts of memory and processing for their main tasks. The Avid iNEWS, for example, demands the MOS Gateway service to be hosted on a computer other than the NCS main servers [18].

Most of the tasks performed on the master control area of a broadcasting facility also demand low latency. For

[1]FIMS allows, but does not require, services to offer a list of MDCF capabilities

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/scc.2016.97

example, if a master switcher (usually an embedded device) is programmed to "play" a playout device when a certain input is selected, it will assume that the playout will respond to the command in a small, predetermined time window. While some protocols allow the client to specify a pre-roll time or frame count, such window is not enough to protect from service response jitter.

Finally, a challenge not unique to broadcasting is service heterogeneity. For example, the play operation can be, at first, seen as a simple universal operation. However, a vendor may not only require additional parameters (such as a pre-roll time), it may also ignore a play command if the device is already playing a scheduled item, or may start playing the next scheduled item. In the absence of a single universal definition for each and all services, for safety, services from different vendors must be considered as incompatible. Efforts for standardization of control protocols are recent and have small and sometimes overlapping scopes. New products while offering some standardized protocols, also present valuable functionality through proprietary protocols.

## IV. Related work

SOA literature is mostly focused on Web Services, and until recently, more specifically SOAP Web Services. However, RESTful services, OSGi services, and service-oriented standards for embedded devices may coexist in a single heterogeneous environment. Similarly, in broadcasting facilities, neither SOAP nor the standardization attempts discussed in section II can be assumed. Three groups of approaches were identified for dealing with heterogeneity in SOA literature, and some representative works are discussed in this section.

The first group contains approaches based on process engines which abstract the heterogeneity. Pautasso and Alonso [19] discuss a framework for managing and executing composite processes with arbitrary non-SOAP services, integrating, in a case study, SOAP Web services, Unix processes and Java code snippets. Lee et al. [20] also present an extended BPEL engine including support for SOAP, REST and OSGi services, as well as Android activities through OSGi adapters.

In the second group, services are created to act as mediators. Battle and Benson [21] propose an SPARQL endpoint for data obtained from SOAP and REST services. Focusing on SoHo devices, Felisberto et al. [22] propose a middleware and service broker allowing transparent interoperability between DLNA[2] and DPWS[3] services through virtual devices that translate requests. Pourezza and Graham [23] use a modified OSGi distribution with OSGi driver services that expose UPnP and Jini services to OSGi and announce OSGi services through the UPnP and Jini discovery protocols.

[2]http://www.dlna.org/
[3]http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01

Finally, in the third group, services are semantically annotated, and interaction with them occurs only at a semantic level. This approach is suggested by the work of Roman et al. [24], where WSMO-Lite is used to semantically annotate descriptions of RESTful and SOAP services.

For the aforementioned works, a direct application to a broadcasting environment would imply in devices being excluded from the integration architecture. From the challenges enumerated in section III, low-latency and constrained resources of the devices are challenges to all listed approaches. The proposed integration architecture uses scripts in an attempt to both introduce small latency and consume a small amount of computational resources.

However, the use of scripts to formulate service compositions is not a novel concept as well. Paluska et al. [25] present a framework for automatically composing adaptive pervasive applications from scripts that implement a goal and may require sub-goals. The approach could be easily adapted to make the generated scripts independent from the planner, avoiding some challenges in section III, but with semantics and data heterogeneity issues still open.

## V. The Integration Architecture for Broadcasting Facilities

The proposed integration architecture is centered on a service repository in which devices (and on-site technicians) publish semantic descriptions of services, their hosting devices, and related code segments. When necessary, usually when its is first set up or reconfigured, the client device sends a service selection request to the repository, describing an ideal service and including some non-functional constraints. Service selection requests are responded with a stub that allows the requesting device to execute the service in a way analogous to calling a local procedure, independently from the repository. This allows the transparent selection of services in a heterogeneous environment, where a baseline protocol, such as SOAP, is not available. The flexibility provided by stubs is such that, a stub composed by the repository may combine Web Services and a service hosted on a device connected to the local serial port, in order to achieve the functionality requested by the repository client.

To cope with a highly heterogeneous environment, both data mediation and service mediation mechanisms are pro-

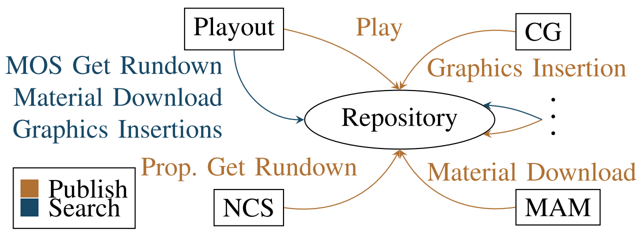


Figure 1. Subset of an example of broadcast facility using the proposed architecture.

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/scc.2016.97

vided. Since only some protocols use XML for data representation, and many use proprietary representations, a lower-level approach based on converter functions is taken. For service mediation, mediating stubs combine available services to offer the functionality described by another service profile. The mediator code itself provides the service, which may be a stateful process involving the services requested by the mediating stub.

Stubs and converters (described in more detail in subsection V-A) are written by the vendor or by on-site technicians as short Lua[4] scripts. Technicians write these scripts in two situations. First to integrate equipment as service providers without cooperation from the vendor. And second, to publish services specific to the broadcaster that are not associated with any equipment. Lua provides a lightweight execution environment, which can easily be embedded in resource-constrained devices that will consume services. To further assist these limited devices, the repository provides ready-to-use composed stubs offering an interface compatible with the proprietary protocols already in use by the devices. Although the requests are based on semantic descriptions, useful requests can be constructed from simple templates of RDF serializations, without the need to actually process the data.

## A. Service Description

Services are described according to an ontology based on OWL-S [3]. To cover concepts and details pertaining to the environment and to the proposed architecture, new classes and properties were introduced. In addition, the prototype does not support WSDL grounding, processes, preconditions and effects, present in the OWL-S ontology. The ontology is split into 5 modules: *addressing.owl* for service endpoints and devices' identifying addresses; *code.owl* for stubs, converters and APIs; *device.owl* for devices; *request.owl* for classes and properties exclusive for selection requests; and *service.owl* that acts as the main module and defines additional service properties.

Services may include the two properties shown in Figure 2, of which *hasEndpoint* is mandatory for the service to be selected. The *Endpoint* class is a lower-level replacement for OWL-S grounding: it holds an address and channel configuration[5], which are used by the stub to communicate with the service.

A large portion of the ontology is devoted to modeling stubs and converters, a subset of which is displayed on Figure 3. *Stub* instances represent fragments of Lua code, which may be used to invoke services with an particular profile at an endpoint of a given type. The Lua code

[4]http://lua.org

[5]UART parameters, such as baud rate and stop bits, cannot be negotiated by the devices. These are usually set by the hosting device which then passively waits for clients.

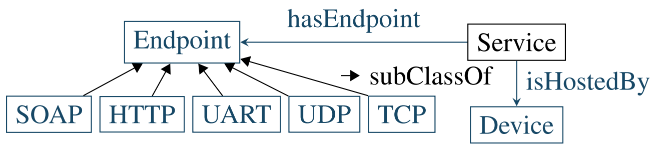


Figure 2. Service description (new elements in light blue)

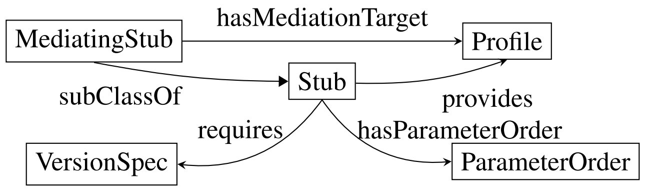


Figure 3. Subset of code.owl

defines an object that receives the service endpoint on the constructor, and invokes the service through a *call* method.

*MediatingStub* instances do not allow access to a remote service, but implement it themselves. Instead of requiring an endpoint, their constructor requires a dictionary of composed stubs providing access to the necessary services. These other services, called mediation targets, are specified through service selection requests[6] as part of the mediator description.

For the repository, to select a service means constructing a tree of *MediatingStub* and *Stub* instances that will allow the client to obtain the results and effects documented on its request. The client receives not the tree, but a ready-to-use composed Lua stub object created from this tree by binding actual services to the stub objects and wiring the inputs and outputs of the selected mediation targets with the *MediatingStub* instances that required their selection. The wiring process involves reordering of parameters (as the Lua stub objects identify parameters based on the order in which they appear in the method/function signature), and the use of converter chains, later discussed, to perform data mediation.

To execute on the repository client's Lua environment, the stub may require some non-standard APIs in a version range. Three version range types are allowed by the ontology: any version, an exact version, or a semantic versioning[7] range. During selection, the APIs required by the stubs are checked against the APIs present at the Lua environment provided by the repository client.

In the absence of a base data representation mechanism such as XML, the parameters to these stubs may require incompatible representations of the same information and no type system supporting down/up-casting and coercion can be defined. Upcasting, also known as type polymorphism in programming languages, is emulated by the use of Lua functions that perform conversion from a more specific representation to a more generic one, possibly discarding

[6]These requests do not allow additional arbitrary SPARQL constraints.

[7]http://semver.org/

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/scc.2016.97

information in the process. These functions are semantically described by the *Upcaster* class. For all other types of data mediation, such as converting between equivalent representations of the same data, Lua functions described by the *DataConverter* class are used. The preservation of information during the conversion performed by this second type of function is assumed only to guide service selection and composition, and is not enforced.

It is not practical for developers and users describing services to provide upcasting and conversion functions to (and from) all useful types. The repository automatically discovers compositions, named converter chains, of these functions to solve data heterogeneity issues that arise when wiring parameters of stub objects.

## B. Service Selection

Functional selection is performed by attempting to select the service or tree of mediators, with minimal dissimilarity to a requested hypothetical profile, that can be used to build a composed stub. This profile includes the I/O parameters and has as its *rdf:type* a functional class, which is a subclass of the OWL-S *Profile* class. The dissimilarity heuristic is a pair $(d_{tax},\ d_{data})$ of taxonomic and data dissimilarity, which can be ordered lexicographically.

The taxonomic dissimilarity between two functional classes is computed as a refinement of the method proposed in [26]. The numeric values 0, 1 and 2 respectively correspond to EXACT, PLUG-IN and SUBSUMES. For PLUG-IN and SUBSUMES, a decimal value smaller than 1 is then added, according to how many *subClassOf* edges separate candidate and request functional classes. Partitions are created when the candidates share the same integer divergence value, and the difference between the fractional values of any two elements on a partition is less than a constant $\rho$. Candidates in the same partition are considered taxonomically equivalent.

To grade the dissimilarity between two distinct data types $a$ and $b$, the least-cost converter chain from $a$ to $b$ is used. Every *DataConverter* in a chain has unitary value, while every sequence of $n$ consecutive *Upcaster* instances has value $2^{n+1}$. The sum of these values yields the chain cost. When the source and target types are the same, no chain is needed and the dissimilarity is zero.

The dissimilarity of the I/O parameters of a candidate profile is defined by the parameter mapping between the request and the candidate profile. Every request input will be mapped to the candidate input requiring least-cost converter chain, the same is done for every candidate output. Once the least-cost chains are selected, the sum of all costs represents the parameter dissimilarity of the candidate profile. However, if a single parameter is selected as the target of more than one source, then the dissimilarity is infinite, given that the mapping is ambiguous.

As discussed in subsection V-A, mediating and regular stubs may be nested in a tree, using converter chains to wire parameters, and bound to specific services. The dissimilarity of such tree is the sum of all individual nodes' values. The composed stub tree is a subtree of a larger, often intractable, tree called selection tree. All candidates with finite dissimilarity for every request posed by every node are present in the selection tree, whose root is the client requested profile. The desired subtree can be found by, recursively, at every choice point, selecting the subtree with smallest dissimilarity.

The challenge, however, is to find such subtree while materializing as little as necessary of the selection tree. Figure 4 shows the top-level algorithm that starts from a single-node tree and controls its exploration by applying operations in breadth-first order and monitoring predicates. Each node has a dynamic working set of children for each request it poses. These sets change as events resulting from node operations propagate upward on the tree, and contain only the best heuristically classified children. The heuristic approximates the least taxonomic dissimilarity of a composed stub tree branch rooted at the evaluated node. The subtree reachable only from the working sets is called the working tree.

```
1: $expandQueue \leftarrow$ ExpandTree($root$)
2: **while** $\neg\, isComposed(root)\ \wedge \neg\, isDeadEnd(root)$ **do**
3:     **if** $isComplete(root)$ **then**
4:         ComposeTree($root$)
5:     **end if**
6:     **if** $\neg\, isComposed(root)$ **then**
7:         **if** $isEmpty(expandQueue)$ **then**
8:             $expandQueue \leftarrow$ ExpandTree($root$)
9:         **else**
10:            $expandQueue \leftarrow$ ExpandTree($expandQueue$)
11:        **end if**
12:    **end if**
13: **end while**
14: **return** $root$ = **null** ? **null** : AssembleStub($root$)
```

Figure 4. Main search control loop

The *expand* operation, when the node working set is empty, computes the lowest, not yet computed candidate taxonomic partition, and adds all the newly discovered child nodes to the tree. In any case, the working set is added to the queue for further expansion. A tree expansion ends when the queue is exhausted (line 10), when the root becomes complete (line 8) or a when a dead end is reached (line 2). The tree is complete when no root or mediating stub node has an empty working set. The parent-child links may be missing parameter mappings, which are computed by the composition operation (line 4) when executed on the child node. When all nodes in the working tree have known mappings, the selection tree is said to be composed, and from it, the composed stub is assembled (line 14).

The solution, however, cannot be said to be optimal: in face of the evaluated scenarios, a non-admissible heuristic

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/scc.2016.97

is used when determining the working sets. The heuristic dissimilarity is a $(P_{tax}, targets, depth)$ triple, where all elements measure the maximal values observed in the working tree: $P_{tax}$ is the representative dissimilarity of the taxonomic partition and is by itself an admissible heuristic; $targets$ is the number of requests posed by the node; $depth$ is the node level compared with the selection tree root. These triples are ordered lexicographically, ignoring $depth$. However, when the $depth$ of two heuristic values differ by more than the value on Table I, the ordering between the $depth$ values prevails over the lexicographical one. This is done to avoid deep exploration of exponentially sized trees, such as those in subsection VI-B, before exploring higher taxonomic dissimilarities.

Table I
DEPTH DIFFERENCE THRESHOLDS FOR DEPTH-BASED ORDERING

| Level | Range | E | H | P | S |
|---|---|---|---|---|---|
| EXACT | $0$ | 1 | | | |
| HALF EXACT | $0.5$ | 2 | 1 | | |
| PLUG-IN | $[1, 2)$ | 4 | 2 | 1 | |
| SUBSUMES | $[2, 3)$ | 6 | 4 | 2 | 1 |

## VI. EVALUATION

To evaluate the feasibility of the proposed architecture, the service repository was implemented in Java 8, using Stardog[8] 4 as the triple store. The repository only uses a single triple store embedded within the application, a scheme that eases deployment and reduces communication overhead, but is neither scalable nor highly available. Interaction with the server is possible through SOAP Web Services, but the automated experiments bypass this layer. The data collected mostly refers to wall-time required for the completion of key tasks in service selection. To avoid interference in the time measurements, before the samples are taken, the garbage collector is run, and system I/O buffers are flushed with the sync Unix utility. Stardog itself, as is usual with databases, shows high query times for newly created databases or newly started servers. To compensate for this, the first samples taken are discarded. Additional countermeasures adopted include using the JVM for a short set of measures and randomizing the measure order.

In the proposed architecture, heterogeneity can be compensated using either data converters or service mediators. These approaches are complementary, but to simplify evaluation, scenarios are simulated in which heterogeneity is completely handled by only one of the two. A plausible situation which stresses the selection algorithm for both approaches is to have a set of protocols (a collection of service profiles and data types), where each provides data or service mediation to a certain number of other protocols.

In the absence of adequate data on the number of vendors present on a same broadcasting facility for a single type of device, semantic service descriptions for this scenario are randomly generated. The MOS protocol is used as inspiration for the random protocols. MOS provides 45 services, which in the SOAP version use the "document" binding style. The generated protocols also have 45 services, but unlike MOS, each service has a return message (in MOS all services respond with *roAck*) and the profile types (functional classes of the services) are distributed in pairs where one is made a subclass of the other.

The test parameters and results for data converters are presented in subsection VI-A, while subsection VI-B discusses the same for services mediators. Both scripts[9] used to collect and process the data presented in the following sections, as well as the prototype source code[10] are available online.

### A. Data Converters

To solve the aforementioned heterogeneous broadcasting facility using data converters, each type of each protocol is mapped with a single data converter to a single type in other protocols as indicated by the $choices$ parameter which varies from 1 to 15 of the total 16 protocols. The types and services of each protocol are isomorphic, the only difference between the elements of different protocols being the namespace of their IRIs. The performed request consists of a profile with the functional class from a (target) protocol, and the parameter types from a (source) protocol. To satisfy the request, a stub must be assembled to receive the types from the source protocol and convert them before invoking the service of the target protocol. An additional variant of this test was also performed, in which the request was such that there were no possible converter chains. To reach this conclusion, the algorithm had to explore a significant portion of the conversion and upcasting graphs.

Table II
STATISTICS FOR CONVERTER-BASED SERVICE SELECTION

| | *Path search* (ms) | | *Remainder* (ms) | |
|---|---|---|---|---|
| **has path** | **min** | **max** | **min** | **max** |
| yes | 0.341 (0.69%) | 1.155 (1.33%) | 49.228 | 86.725 |
| no | 0.023 (0.01%) | 7.286 (2.73%) | 160.981 | 266.516 |

As shown in Table II, since the discovery of the optimal conversion paths uses in-memory graphs, finding the paths (or their absence) in this test is not particularly challenging. The remainder of the selection process includes performing SPARQL queries to obtain the candidate service, the parameter types of the request and the candidate profiles, and the APIs requested by the stub registered for the profile. When the converter path does not exist, additional SPARQL queries need to be performed searching for services at higher taxonomic dissimilarity levels, without success.

[8]http://stardog.com/

[9]https://bitbucket.org/alexishuf/broadcast/

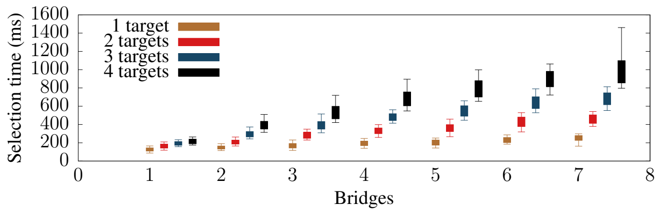
[10]https://bitbucket.org/alexishuf/broadcast-scc-tests/

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/scc.2016.97

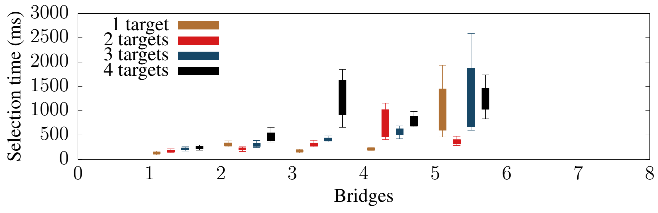
## B. Service mediators

For each profile of each of the 8 protocols, *bridges* mediating stubs that provide the profile using profiles of other protocols are generated in two phases. First, all mediators that require a designated target protocol's profiles are generated. Secondly, all mediators using profiles of all protocols but the designated target, are generated. After the repository is populated, a selection request using a profile of a non-target protocol and requiring the services used by the stub to be hosted on the target protocol's device, is issued.

In the first set of experiments, shown in Figure 5, there is a mediator with taxonomic dissimilarity of 0 that fulfills the request. The number of nodes in the selection tree to be explored, ignoring the non-admissible component of our heuristic, is $O(b\,p)$ where $b$ is the number of bridges per protocol and $p$ the number of protocols (8).



(a) Using non-admissible heuristic



(b) Using only taxonomic partition as an admissible heuristic

Figure 5. Service selection time on a mediating stub scenario. The best composition has a taxonomic dissimilarity of 0

In the tests shown in Figure 5, despite the solution being found with a linear number of nodes explored, the selection tree is sized $O((b\,p)^p)$. If the best match in the repository has a taxonomic divergence of 0.5, without the non-admissible component of the heuristic, $O((b\,p)^p)$ nodes must be explored. The results using only the non-admissible heuristic for this variation are shown in Figure 6.

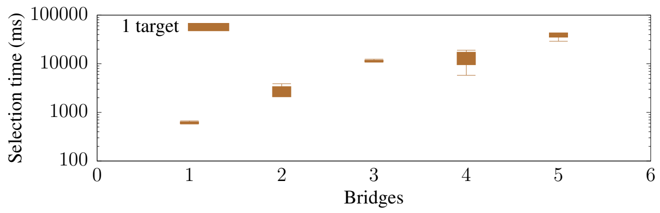


Figure 6. Service selection time on a mediating stub scenario. The best composition has a taxonomic dissimilarity of 0.5

In all test scenarios, the majority of the selection time is spent processing SPARQL queries. For Figure 5a the SPARQL queries amount to an average of 85.6% with $\sigma = 2\%$, and respectively 86.1% and 4.6% for Figure 6. The coefficient of variation for each of these portions is similar, however, the wall time measurement may be vulnerable to background threads controlled by Stardog.

The prototype has no notion of protocols, which are metaphors used only to generate test data. Therefore, all services are considered unrelated, and the non-functional constraint of the device in which selected services must be hosted is treated as an opaque SPARQL group pattern. Due to the use of SPARQL to express non-functional constraints, and richness of the RDF service descriptions, it is not viable to create a proprietary in-memory graph to manage all data. Moreover, the prototype does not exploit the fact that all functional aspects of service selection, centered on the service profiles, are independent from the actual services and devices found at a broadcasting facility.

## VII. Conclusions and Future Work

This paper presented a SOA-based solution for device integration in broadcasting production facilities. In tackling the challenges of such technologically heterogeneous environment, an approach building on OWL-S and Lua code composition was taken. While this is not the first attempt to apply SOA in broadcasting, no documented attempt to use semantic Web Services or automated service composition in this environment was found. The present work also attempts to better integrate the Web and the non-Web services, which are the majority in the environment, instead of setting up bridge Web Services. Despite our focus on broadcasting facilities, some ideas evaluated in the present work, could be useful in other environments without a single base protocol, such as small office and home automation, or professional environments with legacy software and hardware.

While no appropriate quantitative data could be obtained to estimate an acceptable time limit for stub composition, the experimental results hint the time required for a device to obtain stubs for all the services it will invoke, could be acceptable for converter-based integration but inconveniently high for mediator-based integration. There are two common situations in which a device would query the repository for a stub: when it is set up in the broadcaster's studio, and when one of the services used by the device becomes faulty or unavailable. A future work for the later situation is to monitor QoS parameters and faults of the services used by a stub (the repository could perform the monitoring for constrained devices) and use a simpler algorithm to find replacements only for faulty components.

Being a prototype, aspects such as security and availability were not addressed. Any attacker with access to the local network may use the repository to remotely inject code on the broadcaster devices. As for availability, the repository is a monolithic entity and can be a single point of failure.

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/scc.2016.97

Compared with related works in the broadcasting industry and in SOA literature, our prototype is positioned as a complementary approach. Lua stubs were the approach that covered more ground considering the challenges identified, but a hypothetical commercial product could include ideas from many of the related works. For example, stubs could be generated from semantically annotated WSDL files and converters from XSD and XSLT files. Also, service compositions created by the repository could be made available through MDCF and DPWS discovery mechanisms by virtual devices. Finally, the broadcaster could define his/her process using abstract services, which would be selected from the repository and executed by an extended BPEL engine.

## References

[1] S. Pizzi and G. Jones, *A Broadcast Engineering Tutorial for Non-Engineers*. Taylor & Francis Group, 2014.

[2] E. Tozer, Ed., *Broadcast Engineer's Reference Book*, 1st ed. Focal Press, 2004.

[3] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, *OWL-S: Semantic Markup for Web Services*, SRI International Std., Mar. 2008. [Online]. Available: http://www.ai.sri.com/daml/services/owl-s/1.2/

[4] J. Berryhill, "Automation applied to television master control and film room," *IRE Trans. Broadcast Transmission Systems*, vol. PGBTS-9, no. 1, pp. 11–20, Dec. 1957.

[5] K. Paulsen, *Moving Media Storage Technologies: Applications & Workflows for Video and Media Server Platforms*. Taylor & Francis, 2012, ch. 3.

[6] "Media object server protocol v2.8.4," MOS Development Group, 2011. [Online]. Available: http://mosprotocol.com/wp-content/uploads/2014/04/MOS-Protocol-2.8.4-Current.htm

[7] *ST 377-1:2011. Material Exchange Format (MXF) – File Format Specification*, Society of Motion Picture and Television Engineers Std., Jun. 2011.

[8] *ST 2021-1:2015. Broadcast Exchange Format (BXF) – Requirements and Informative Notes*, Society of Motion Picture and Television Engineers Std., Dec. 2015.

[9] *ST 2071-1:2014. Media Device Control Framework (MDCF)*, Society of Motion Picture and Television Engineers Std., May 2014.

[10] D. M. Weise, "Computerized Techniques for Complete Television Station Automation," *IEEE Trans. Broadcast.*, vol. BC-14, no. 4, pp. 151–160, 1968.

[11] M. A. Pusateri, "Living with Video Servers in a Digital Broadcast Facility," in *SMPTE Technical Conf. and Exhibit, 140th*, 1998, pp. 1–9.

[12] S. Rodd, "Linking broadcast and business systems," in *Int. Broadcasting Conv.*, 1997, pp. 101–105.

[13] P. J. Lude, "The fully integrated multichannel broadcast system: harder than it looks," in *Int. Broadcasting Conv.*, 1997, pp. LP16–LP21.

[14] M. Hayashi, N. Yoshihara, S. Hosoi, T. Umiuchi, and K. Hara, "An Approach to the Automation of Television Studio Program Production," *Journal of the SMPTE*, vol. 73, no. 11, pp. 942–946, 1964.

[15] *Specification of the FIMS media SOA framework*, European Broadcasting Union Std., Oct. 2015.

[16] "Media object server protocol v3.8.4," MOS Development Group, 2011. [Online]. Available: http://mosprotocol.com/wp-content/uploads/2014/02/MOS-Protocol-3.8.4-Current.htm

[17] (2015, Jul.) AP ENPS integration guide. Associated Press. [Online]. Available: http://www.enps.com/pages/integration/integration_guide

[18] *iNEWS MOS Gateway – Version 4.2.1 ReadMe*, Avid, Dec. 2015. [Online]. Available: http://resources.avid.com/SupportFiles/attach/Broadcast/MOSGWv4.2.1-ReadMe.pdf

[19] C. Pautasso and G. Alonso, "From web service composition to megaprogramming," in *Technologies for E-Services*. Springer, 2004, pp. 39–53.

[20] J. Lee, S.-j. Lee, and P.-f. Wang, "A Framework for Composing SOAP, Non-SOAP and Non-Web Services," *IEEE Trans. Serv. Comput.*, vol. 8, no. 2, pp. 240–250, Mar. 2015.

[21] R. Battle and E. Benson, "Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)," *Web Semantics: Sci., Services and Agents on the World Wide Web*, vol. 6, no. 1, pp. 61–69, Feb. 2008.

[22] T. Z. Felisberto, E. D. Tramontin, F. d. C. d. Santos, A. S. Morales, F. Siqueira, and G. M. d. Araújo, "UDP4US: Universal device pipe for ubiquitous services," in *2015 Brazilian Symp. on Computing Systems Engineering (SBESC)*, Nov. 2015, pp. 36–41.

[23] H. Pourreza and P. Graham, "On the fly service composition for local interaction environments," in *Pervasive Computing and Commun. Workshops, 2006. PerCom Workshops 2006. 4th Ann. IEEE Int'l Conf. on*, 2006, pp. 6 pp.–399.

[24] D. Roman, J. Kopecký, T. Vitvar, J. Domingue, and D. Fensel, "WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs," *Web Semantics: Sci., Services and Agents on the World Wide Web*, vol. 31, pp. 39–58, Mar. 2015.

[25] J. Mazzola Paluska, H. Pham, U. Saif, G. Chau, C. Terman, and S. Ward, "Structured decomposition of adaptive applications," *Pervasive and Mobile Computing*, vol. 4, no. 6, pp. 791–806, Dec. 2008.

[26] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *Int. Semantic Web Conf. ISWC 2002*, ser. Lecture Notes in Computer Science, I. Horrocks and J. Hendler, Eds., vol. 2342. Springer Berlin Heidelberg, 2002, pp. 333–347.

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/scc.2016.97