

A TRANSLATION ENGINE IN SUPPORT OF CONTEXT-LEVEL INTEROPERABILITY

KYM J. POHL

CDM Technologies Inc.
San Luis Obispo, CA 93401, USA
805-541-3750 / Fax: 805-541-8296 / kpohl@cdmtech.com

Abstract: The support of context-level interoperability demands increasing attention in today's arena of semantics-oriented decision-support systems. Unlike data-oriented exchange, such semantic interoperability must venture beyond the elementary communication of discrete data values and endeavor to translate between significantly more expressive, context-rich representations. Further, support of this level of interoperability must not require a contamination of the native perspective embedded within each participant's representation. Offered as a foundation upon which specific interoperability solutions can be developed, this paper presents a service-oriented framework supporting an extensible set of translation paradigms to effectively connect expressive ontology-based environments. Fundamental to this capability is the notion of a remote service request. Employing such a metaphor as the basis for participant interaction allows each system within a universe of potentially diverse representations to interoperate as collections of invocable services. By transparently marshalling such requests between client and service representations, each member of this multilingual reality is empowered to interoperate within the familiar confines of its native representation. This paper concludes by evaluating this framework in terms of the Levels of Conceptual Interoperability Model (LCIM).

Keywords: interoperability, decision-support, semantic mapping, LCIM

1. THE INTEROPERABILITY DILEMMA

Semantic interoperability is increasingly gaining significance in today's information age. With the context-sensitive analytical demands placed on today's software-based decision-support systems [Pohl et al, 1997] it is more critical than ever for interaction among such systems to surpass the simple sharing of data and venture into the realm of semantic exchange. Context-oriented – as opposed to data-centric – systems rely heavily on the contextual depth of the descriptions over which they operate. Contextual depth and semantic expression (i.e., property and relationship-rich descriptions of the entities and concepts representing a relevant set of domains), fundamental to context-level systems, enable meaningful assistance in the areas of resource-allocation, threat-analysis, and conflict-resolution – to name a few. Whether interacting with other context-enabled systems or accepting data feeds from more data-centric contributors, the context-enabled system needs to communicate context as opposed to simply *structure*, which no doubt places a significantly burden on the representational depth of the overall exchange. Providing support for such context-centric interoperability is currently a highly researched topic within academia and industry [Karsai, 2000].

Perhaps not surprisingly, it is the fundamental strength of context-centric decision-support systems that presents the most challenging obstacle in this endeavor. This critical

enabler, and simultaneous nemesis, is *representational expressiveness*. As the term implies, the context-oriented approach to the development of decision-support systems endeavors to transcend the classical data-centric approach to representation (i.e., isolated chunks of typically numeric or string-based data with few or no inter-relationships, essentially devoid of any derivable meaning) and incorporate the potentially numerous relationships, constraints, and business rules that are needed for the more complex analysis inherent in agent-based, decision-support environments. A critical aspect of such representational depth is *perspective*. Biases associated with the way a particular entity or concept is *viewed* prove exceedingly significant to the decision-making process. As such, perspective is a critical ingredient to truly effective context-oriented representation. When modeling a fundamental aspect of a transportation domain, for example, supporting the perception of a *vehicle* as a means for transporting cargo may be quite a useful perspective to maintain. However, for manufacturing operations, this same vehicle is perhaps more effectively represented as a sequenced collection of assembly stages. Preserving such representational individuality can quickly lead to gross disparities among interoperating systems [Pohl J., 2001; Pohl K., 2001].

To support meaningful collaboration within such potentially diverse environments, the incorporation of a translational component capable of mapping between expressive contexts proves useful, but only when

accomplished in a manner that transparently preserves the perspectives native to individual users. This focus drove the development of the capability presented in this paper. Neither a complete nor universal solution, this capability is designed as a *framework* upon which specific interoperability solutions can be developed.

The following sections describe the criteria used in developing this focused capability, the various technologies employed to realize its implementation, the architecture that combines these elements, and finally, the interoperability platform as assessed in terms of the Levels of Conceptual Interoperability Model (LCIM) [Tolk and Muguira, 2003].

2. CRITERIA

To successfully address these issues, candidate technologies must satisfy several criteria discussed below.

Flexibility constitutes a primary goal of any solution intended for repeated application to varying interoperability scenarios. Such adaptability requires the clear separation of framework from application specifics. In other words, a candidate framework should support the various abstractions associated with translation-based interoperability among heterogeneous representations. Some of these abstracted concepts include the notion of translation itself, which should remain broad enough to include an extendable variety of translation paradigms.

Further, this capability should identify and formalize the necessary interfaces to support adaptation of native client connectivity to the paradigm provided by the bridging framework. This entails addressing the necessary functionality and blueprint to effectively *adapt* client systems to the specific interoperability framework and paradigm offered within this environment.

Another significant property of any capability realistically intended for broad application remains the support and promotion of industry standards. This proves particularly significant when a high degree of reusability is intended. Accordingly, such a facility should center on industry-familiar technologies, standards, and tools. Adherence to available standards not only offers a helpful guide in developing a particular capability but also aids in averting the unpleasant consequences associated with reworking a problem to fit an inflexible implementation. This requires particular attention in a field where complexity and *one-off* solutions abound.

As stated in the previous section, a critical ingredient for effective interoperability among context-oriented systems is the preservation of native perspective. Successful addressing of this issue requires the ability to understand

the subtleties inherent in such a concept (e.g., implied domain-specific constraints that have equally obscure and individual counterparts when considered from other domain-related perspectives). The considerations involved in translating between such perspectives often require a level of reasoning approaching the realm of expert systems [Giarratano and Riley, 2003]. In many respects, for the more complex context-oriented interaction, a level of decision-support on par with solutions supporting multi-variable, complex problems is needed. In fact, effective contextual translation may in certain cases require the participation of the human decision maker for representing the higher-level logic not readily encodable even within today's cutting edge intelligent software.

A final important, yet often overlooked, criterion for successfully addressing this interoperability problem requires a focus on the exchange of expressive content that can nonetheless sufficiently support less complex exchanges without incurring the overhead associated with support for interoperability in its more complex form described above. This often occurs when capabilities targeting complex problems offer excessively engineered and subsequently inefficient solutions for fairly straightforward scenarios. The goal here is to support a range of complexities and to limit any incurred overhead whenever possible.

3. SUITABLE TECHNOLOGIES AND DESIGN PATTERNS

To successfully address the challenging criteria set forth above, several enabling technologies and approaches were investigated. First among these is a Service-Oriented Architecture (SOA) for operation in both standalone and web-enabled forms [Friedman-Hill, 2003; Ewalt, 2002; Graham et al, 2001; Heflin et al; 2002; Hendler et al, 2002; Horrocks, 2002]. In this manner, services can be registered, consequently discovered, and eventually invoked via standard registry lookup and interaction protocols. Further, by defining the functional topology in terms of invocable services, the classical notion of system boundaries can be replaced with the more dynamic notion of composable services effectively expanding and contracting system boundaries as needed.

The eXtensible Markup Language (XML) [Hendler et al, 2002; Hunter et al, 2000] together with its eXtensible Stylesheet Language Transform (XSLT) [Hunter et al, 2000] language counterpart are two additional technologies applicable to the domain of translation. XML provides a flexible means of defining structure through the use of syntactical tags. XML schemas can be developed to describe the structural aspects of entities, notions, and concepts. Receivers can process XML statements based on these schemas in an interpretational manner. The result, although accompanied by a

significant caveat, is a means whereby software components can process incoming content based on a previously unknown representation. However, it should be noted that such discovery is limited to structural characteristics and does not include the discovery of semantics, *or meaning*, vital in context-oriented exchange. Even considering a meta-level schema describing the domains of concepts, rules, and implications, there is still the requirement for processors of such content to be equipped with a pre-conceived understanding of the semantics of even these abstracted domains. At some point, the semantics need to be adequately represented beyond simple structure. That said, however, structural detection does play a significant role in the eventual goal of true contextual discovery but should be viewed with a strong distinction between structure and semantics.

Perhaps one of the more significant additions to the world of tag-based schema languages is the ability to formally describe inter-schema transformation rules in an XML-friendly syntax. XSLT is a language that can be used to describe exactly how content based on one XML schema can be mapped into another. Whether defined statically or more dynamically at runtime, XSLT transforms can be effectively applied for straightforward property-to-property translations.

Although translation at this level is useful, for the more complex transformation inherent in context-oriented representations a more powerful paradigm is required. More extensive reasoning capabilities can be realized through the use of inference engine-based technology. Similar to XSLT, inference engine-based translation represents transformation logic as sets of managed rules. However, as opposed to the aforementioned XSLT mappings, these rule sets can embody significantly more complex logic supporting extensive condition-based pattern matching and context-based inferencing. Some examples of rule-based inference engines include the

CLIPS expert system shell developed by NASA [NASA, 1992] and a somewhat similar, although more easily embeddable rendition within Java-based environments, JESS inference engine developed by Sandia Laboratories [Friedman-Hill, 2003]. In either case, complex transformation logic can be implemented as pseudo-expert systems applying various levels of reasoning to determine the appropriate transformation(s) to apply given an expressive contextual fragment. This added level of sophistication is particularly useful where translated subject matter is bound by different constraints as it moves between representations. Under these circumstances, enforcement of such constraints may require a level of decision-support capable of reorganizing the particular subject matter in a means that produces the appropriate state upon insertion into the target representation.

4. THE INTEROPERABILITY BRIDGE

The capability presented in this paper takes the form of a configurable, service-oriented interoperability framework able to support an extensible set of translation paradigms. Further, the Interoperability Bridge, or *Bridge* for short, employs an overarching interaction model founded on a *remote service request* metaphor. The orientation toward this type of exchange model allows each interoperating system to view its counterparts as collections of composable services invocable using *language* native to the caller. The resulting interoperability environment promotes a decoupled architecture requiring no pre-conceived notion of participant identity other than the invocable services a client chooses to expose.

The main Bridge architecture comprises three components. These include the Service Center, an extensible Translator Pool, and individualized Client Connectors. The following section discusses each of these components in more detail.

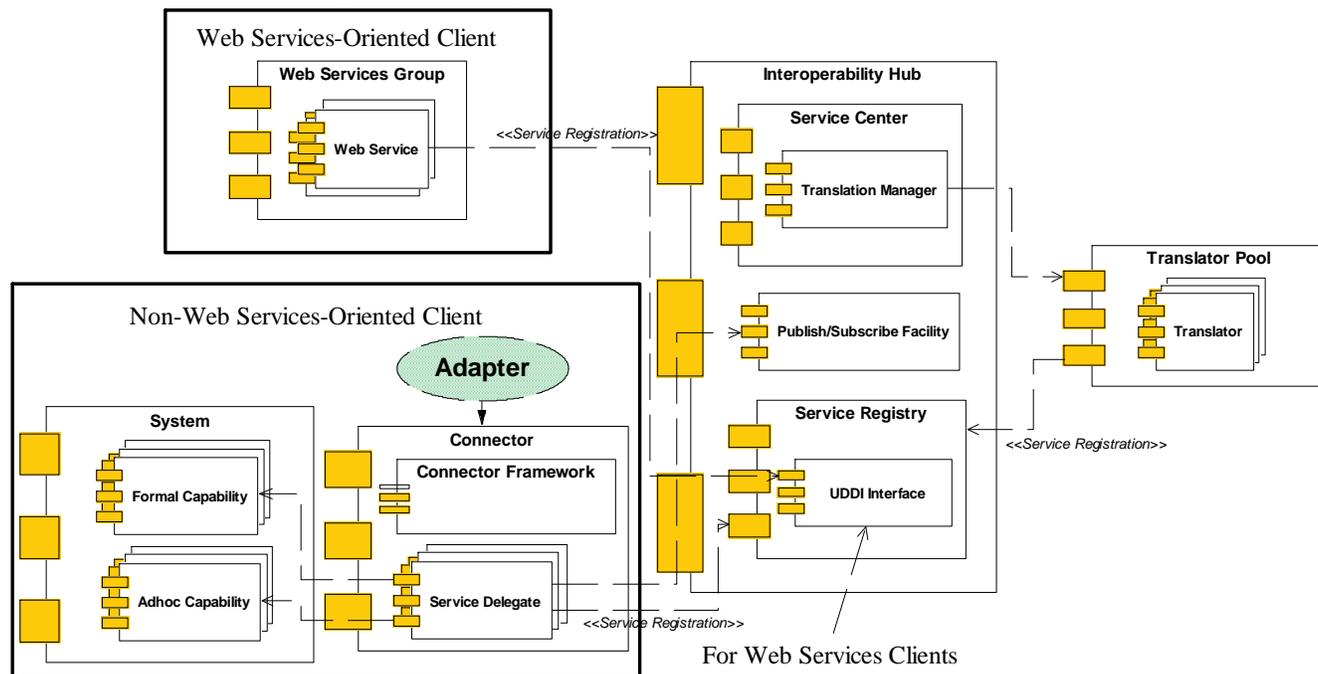


Figure 1 – The Interoperability Bridge Component Architecture

The Service Center

The Service Center provides the overall management required to orchestrate the reception and processing of client requests. As the main interface for Bridge clients, this component is exposed in both a standalone and Web service manner. When deployed as a Web service, the Bridge adheres to the standard Web service design pattern [Graham et al, 2001]. In its standalone form, the Service Center is exposed as a programmatic interface. Figure 1 provides a Unified Modeling Language (UML) [Fowler and Scott, 1997] illustration of the Service Center, in this case configured as a Web service, as it promotes a service-oriented architecture allowing clients to expose both formalized and ad hoc capabilities as composable services available for invocation.

Apart from providing the main client interface of the Bridge, the Service Center also acts as a request broker effectively routing inter-client communication (i.e., service requests and subsequent responses) to the appropriate member of the Translator Pool for effective inter-dialect mapping (i.e., mapping between client representations). The Service Center accomplishes this task in the form of a Translation Manager. The Translation Manager embedded within the Service Center is responsible for discovering and engaging suitable translation services (i.e., translators) to perform the required representational mappings. To further support the Web service interaction model promoted at the main client interface level, the Translation Manager is capable of interacting with registered translators in either a standalone or Web service form.

The Translator Pool

Collectively known as the Translator Pool, and the second of the three primary components comprising the Bridge architecture, the particular set of translators registered with the Bridge at any point in time determines the extent of translation paradigms available to interoperating clients. Further, the breadth and depth of interoperability available within these paradigms is determined by the various translation rule sets with which these translators are equipped. As with membership within the Translator Pool itself, such configuration occurs in a dynamic fashion as translation needs change and desirable rule sets become available. As a translator's scope of language mappings change (i.e., by being equipped with additional or modified mapping rules), such translation capability is communicated to the Translation Manager in terms of both the source and target XML Schema Definition (XSD) that the associated rule set can map between.

The Client Connector

The various Client Connectors that effectively *connect* each client to the Bridge complete the primary Bridge architecture. Each client intending to invoke the services native to the Bridge (i.e., publication, query, subscription) or the remote services offered by other Bridge clients does so via its native Client Connector. Based on a reusable framework provided as part of the Bridge development and execution package, a Connector is built for each type of candidate bridge client in accordance with that particular connectivity model. In essence, these connectors effectively *adapt* a client to seamlessly

operate in terms of the interaction model promoted by the Bridge. This approach has a very significant advantage over more intrusive connectivity models. Essentially acting as a *proxy* for its specific client, the Connector effectively isolates functionality native to the client from any knowledge of the Bridge itself. During invocation of a client's service, the receiving client's Connector processes the request into a series of native function calls. For outgoing requests, clients have a choice as to their level of awareness of the Bridge. For client's wishing to initiate remote requests, such invocation logic may be woven into the functional fabric of the particular client, or it may be solely isolated within the implementation of the Connector, thus preserving a client's decoupled nature. The latter of these two configuration extremes exploits the *delegate* design pattern implemented within the Connector architecture. For more isolated clients, such delegates can react to locally occurring events, issuing remote service requests on an as needed basis. In either case, the results of inter-client service invocations can be asynchronously returned as service requests directed toward the issuer's particular publication service, in whatever form that capability may be.

The Client Connector Framework

To assist in coping with differing models of client connectivity, the Interoperability Bridge platform offers a framework for building and managing Client Connectors. This framework consists of well-defined interfaces, functional modules, and a formalized design pattern for integrating these components with client-specific behavior. Figure 2 illustrates the architecture of the Connector Framework along with the various interfaces on which client-specific concerns are implemented. To provide a better understanding of these Connectors as they operate in conjunction with both client and Bridge functionality, the following section briefly illustrates the Connector workflow involved in processing a typical client service request.

In reference to the Connector architecture presented in Figure 2, outgoing communications are managed collectively by the Export Manager, Export Adapter, and Export Formatter. The main function of the Export Adapter is to employ the client's native mechanism for the notification and subsequent acquisition of relevant events (i.e., remote service requests or, in the case of a less Bridge-aware client, the local events that would indirectly trigger the Connector to invoke a remote

service). In the case of a more isolated client, a common mechanism to facilitate indirect invocation may be some sort of local event service capable of notifying interested parties – in this case the Export Adapter – of certain events. For more Bridge-aware clients, such a mechanism may take the more direct form of explicit method calls issued to an extended Export Adapter implementation.

Having received the outgoing content, the Export Adapter passes it to the Export Manager for processing, which involves managing the necessary reformatting of the communication content into its XML equivalent. Recall that all direct interaction with the Service Center is XML-based. The details of this reformatting operation are completely encapsulated inside the client-specific implementation of the Export Formatter interface. While Service Center clients already capable of communicating in XML can avoid the overhead associated with this extra step, having such a reformatting facility allows non-XML clients to effectively utilize the Bridge in an architecturally organized manner. This again illustrates an underlying theme of this capability to limit constraints placed on system representation and format. Once the communication has been appropriately formatted into its XML equivalent, the Export Manager passes the content to the Service Center for brokering to the appropriate member of the Translator Pool. Following this translation, the content is routed as incoming subject matter to the appropriate Service Delegate implementation of the target client's Connector via the bridge's internal publish/subscribe facility. Adhering to the interface specified by the Connector Framework, each Service Delegate implementation essentially represents a *proxy*, or representative, for a local capability exposed to the remote world. It should be noted that by the time any incoming content reaches a particular Service Delegate, the Service Center, via its Translation Manager, has already performed any necessary representational transformation ensuring that the target connector only receives content compliant with its client's native representation. Once a Service Delegate receives a request, it is passed through the Import Formatter that converts the XML-based content to the target client's native format. Similar to the applicability of the Export Formatter, this step is only necessary if the native interface presented by the receiving client is not XML. It should also be noted that both the Export and Import Formatters do not perform the type of representational transformation undertaken by the translators housed within the Translator Pool.

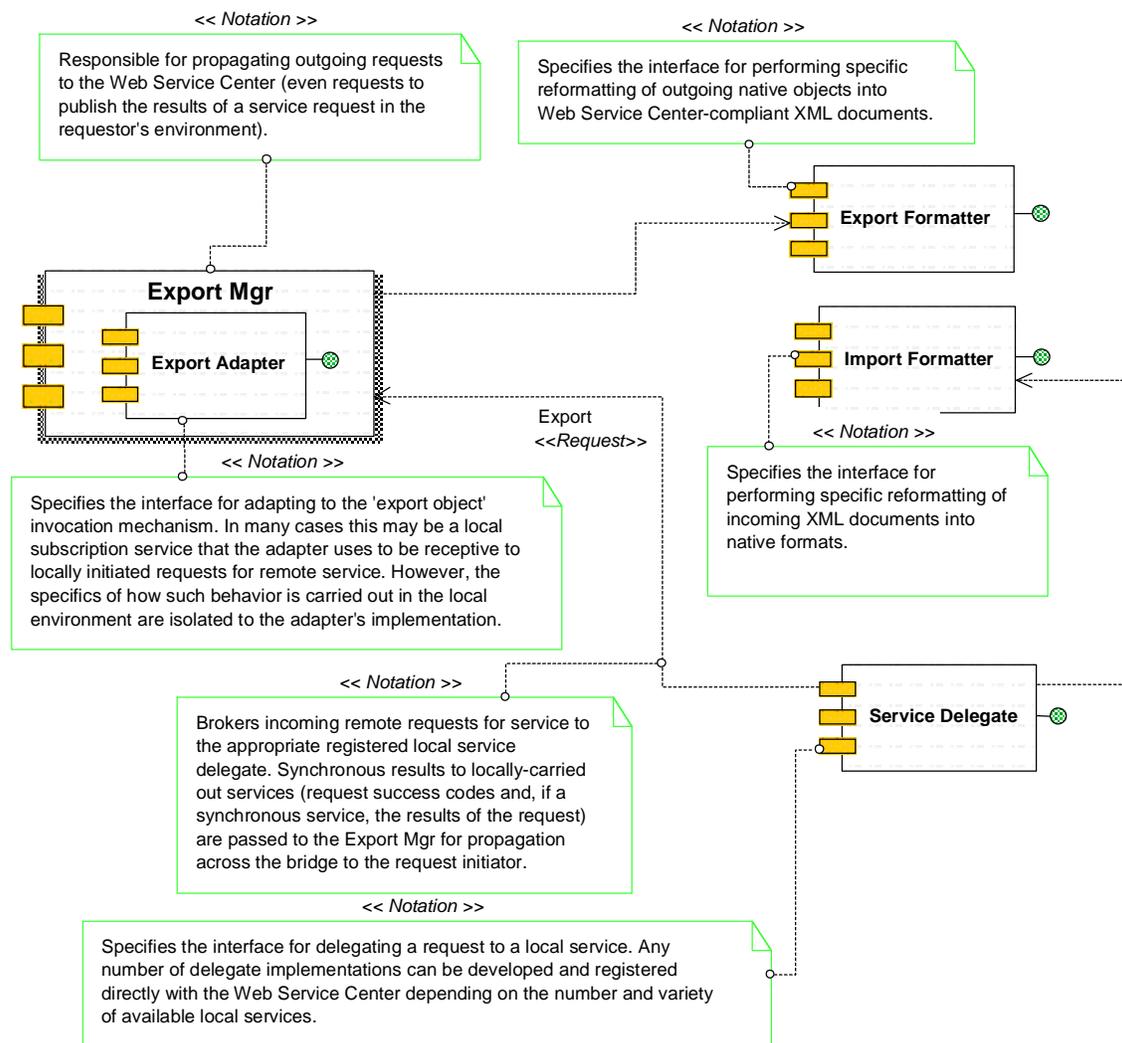


Figure 2 – Connector Framework Architecture

Once the request has been converted into the appropriate format, the Service Delegate invokes the native capability offered by its client to perform the requested service and manages the subsequent returning of any results to the Service Center as outgoing content. Details concerning invocation of and interaction with local capabilities are fully encapsulated within the client-specific implementation of the Connector's Service Delegate interface. Such invocation may take a variety of forms, including direct interaction or creation of a local event that indirectly triggers the desired client functionality. Regardless of the means of invocation, the functionality being requested may exist at varying stages of formality. In other words, since the Service Delegates essentially function as the representatives of an exposed capability, the specific local functionality that constitutes the particular service is encapsulated within the delegate itself and may be in varying degrees of formalization. This proves particularly useful when adapting loosely organized functionality to a more formalized, service-oriented interoperability environment.

The scenario presented above is most suitable where there is no native concept of interoperability outside the scope of a pre-determined set of systems. However, in the case where native capabilities are, in fact, designed to operate in terms of an extensible, open architecture, the role of the Service Delegates can be reduced to managing the reformatting of communications for non-XML systems or else, in the case where XML is supported, omitted completely. While in the latter scenario the native capability would manage its own exposure to the Service Center, it may still benefit from the virtual homogeneous environment promoted by the Interoperability Bridge facility (i.e., each Bridge client is provided the impression that its native language is the interoperability standard when in fact, each client is communicating in terms of a distinct dialect).

5. THE INTEROPERABILITY BRIDGE AND SUPPORTABLE LEVELS OF CONCEPTUAL INTEROPERABILITY (LCIM)

LCIM offers a multi-tiered set of criteria for assessing the degree of interoperability achieved between software systems. Evolving from efforts in the world of modeling and simulation, LCIM presents the following 7-tier interoperability assessment model [Tolk and Muguira, 2003]:

- **Level 0:** Standalone systems having essentially no interoperability.
- **Level 1:** Achieving *technical interoperability*, whereby a communication protocol exists for exchanging data between participating systems. Further, a communication infrastructure is established to enable exchange of bits and bytes with underlying network and communication protocols unambiguously defined.
- **Level 2:** Achieving *syntactic interoperability*, whereby a common structure enables the exchange of information (i.e., a common data format is applied). At this level, a common protocol to structure the data is used with the format of the information exchange unambiguously defined.
- **Level 3:** Applying a common information exchange reference model accomplishes a level of *semantic interoperability*. At this level, the meaning of the data is shared and the content of the information exchange requests are unambiguously defined.
- **Level 4:** *Pragmatic interoperability* is achieved when the interoperating systems are aware of the methods and procedures that each participant is employing. In other words, the use of the data, or the context of its application, is effectively *understood* by the participating systems. Further, the context in which the information is exchanged is also unambiguously defined.
- **Level 5:** A level of *dynamic interoperability* is achieved when interoperating systems are able to comprehend the various changes in state (i.e., assumptions and constraints) made by each participant as execution progresses. Achieving this level of interoperability allows participating systems to effectively exploit such knowledge to their benefit.
- **Level 6:** *Conceptual interoperability*, the highest level of LCIM interoperability, is achieved when the conceptual model (i.e. the assumptions and constraints of the purposeful abstraction of reality) becomes semantically aligned. Further, this level of interoperability requires that conceptual models be

documented in their implementation-independent form and subsequently made available to engineers.

As opposed to an actual instance of a particular representational model, the Interoperability Bridge discussed in this paper offers a platform where such models can be defined and executed. As such, LCIM assessment is performed in terms of the various paradigms the Bridge is designed to support.

Whether deployed in its standalone or Web service form, the Interoperability Bridge provides an underlying communication infrastructure that, through customization of the adaptive Connectors, effectively connects both similar and disparate clients. As such, this capability provides the *technical interoperability* specified in LCIM Level 1.

Both *syntactic* and *semantic interoperability*, LCIM Levels 2 and 3 respectively, are achieved once the Interoperability Bridge is equipped with the appropriate mapping models that effectively tie together client representations. As described earlier, the Bridge provides an expandable set of translation paradigms suitable for both simple and complex inter-representational mapping. The Bridge also provides a pattern for both constructing and integrating additional translation engines thus allowing for incorporation of emerging mapping technologies as they are developed.

LCIM Level 4, *pragmatic interoperability*, requires exposure of client capabilities to other interoperating clients. The Interoperability Bridge supports this task by defining client interaction in terms of a *remote service request* metaphor. In this manner, clients are not only able to expose available functionality to interested clients but can present such capabilities as *conceptual* services that in actuality may be transparently comprised of various collections of formalized or more ad hoc native functionality.

Considering the degree of expressive interoperability manageable by the Bridge, cross-client awareness and comprehension of each other's changes in state (i.e., LCIM Level 5) is, in fact, supportable. However, the specific degree of support essentially depends on the extent to which such aspects are represented in the particular interoperability model the Bridge is configured to manage. In other words, while the Bridge has no native specification for such inter-client awareness, it nonetheless does present a suitable environment to support an interoperability model equipped to represent and convey such notions across its clientele. As mentioned at the beginning of this section, as opposed to a specific instance of an interoperability solution, the Bridge offers a platform enabling varying degrees of inter-client exchange constrained only by the scope and sophistication of the configured interoperability model and the ability of its clientele to process such representation. Further, in support of the higher levels of

client awareness and comprehension associated with LCIM Level 5, the Bridge provides a service-oriented client interaction model together with an extendable set of translation paradigms suitable for managing notions as sophisticated as domain-specific assumptions and constraints.

Although the Bridge is currently only concerned with the interoperability model in its implementation form, it would be interesting to consider the potential for supporting such a model in the more conceptual form specified in LCIM Level 6. Although the Bridge can currently process models specified in terms of the Unified Modeling Language (UML) methodology, the Bridge makes no distinction as to the conceptual nature of such descriptions. In other words, such models are processed by the Bridge as the literal language for inter-client exchange and not the more abstracted form on which various implementations of such a model could be based. To not only acknowledge such a distinction, but to also provide a meaningful level of support, an ability to relate implementation model fragments to their conceptual model counterparts would appear to be helpful. Once such a relationship is understood, there may be potential for the Bridge to exploit this knowledge to automatically determine relationships existing between various conceptual model implementations. In this manner, there would be no need to develop and consequentially configure the Bridge with explicit mappings between implementation models. Such connections would be developed and dynamically maintained by the Bridge at run time. Taking this capability to the next level, if the Bridge could, in fact, comprehend the conceptual nature of an interoperability model, perhaps in terms of a native *concept* ontology, it would appear plausible that such awareness could begin to allow the Bridge to elevate its analysis of client models to their conceptual form. Such capability would set the stage for focusing the Bridge's support for dynamically determining cross-client interoperability on the conceptual level, drawing concept-based relationships between native client models in their conceptual form.

6. CONCLUSION

The Interoperability Bridge offers an effective means whereby existing, perhaps loosely defined, application-level functionality can be adapted to operate within a dynamic interoperability paradigm. Through the use of Service Delegates, the details associated with directly interfacing with local system capabilities are encapsulated within client-specific code fragments and effectively isolated from reusable framework components. With flexibility as a fundamental theme, systems developed with such service-oriented concepts more integrated within their native design are able to avoid any undue overhead associated with such adaptation and more readily exploit the functionality offered by the Interoperability Bridge.

The framework-level approach to interoperability among expressive environments presented in this paper goes beyond traditional connective architectures by addressing the dramatic representational disparity typically exhibited by context-oriented systems. Rather than constrain interoperating environments to a common, albeit perhaps expressively rich, representation, the Interoperability Bridge provides a platform whereby potentially disparate representations can be effectively integrated to form a virtually homogeneous *view of the world*. As a result, interoperating systems can function within a cohesive service-oriented paradigm while still maintaining the native perspectives critical to effective context-based decision-support.

REFERENCES

- Cagle, K., M. Corning, J. Diamond, T. Duynstee, O. Gudmundsson, M. Mason, J. Pinnock, P. Spencer, J. Tang, A. Watt, J. Jirat, P. Tchistopolskii, and J. Tennison, "Professional XSL", Wrox Press Ltd., Birmingham, UK., 2001.
- Daconta, M., L. Obrst and K. Smith, "The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management", Wiley, Indianapolis, IN., 2003.
- Ewalt, D., "The Next Web", Information Week, October 2002, (www.informationweek.com/story/IWK20021010S0016).
- Fowler, M. and K. Scott, "UML Distilled: Applying the Standard Object Modeling Language", Addison-Wesley, Reading, Massachusetts, 1997.
- Friedman-Hill, E., "JESS In Action: Rule-Based Systems in Java", Manning Publications Co., Greenwich, CT, 2003.
- Giarratano, J. and Riley G., "Expert Systems: Principles and Programming", 2nd Edition, PWS Publishing Company, Boston, MA.
- Gil, Y. and V. Ratnakar, "Markup Languages: Comparison and Examples", Information Sciences Institute, University of Southern California, TRELIS project, 2002, (www.isi.edu/expect/web/semanticweb/comparison.html).
- Graham, S., S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama, "Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI", Sams Publishing, Indianapolis, IN, December 2001

Heflin, J., R. Volz and J. Dale (eds.), "Requirements for a Web Ontology Language", W3C Working Draft, July 8, 2002, (www.w3.org/TR/webont-req).

Hendler J., T. Berners-Lee and E. Miller, "Integrating Applications on the Semantic Web", Journal of the Institute of Electrical Engineers of Japan, 122(10), October 2002, (pp.676-680).

Horrocks, I., "DAML+OIL: A Description Language for the Semantic Web", IEEE Intelligent Systems, Trends and Controversies., 2002

Hunter, D., C. Cagle, D. Gibbons, N. Ozu, J. Pinnock, and P. Spencer, "Beginning XML", Wrox Press Ltd., Birmingham, UK., 2000.

Karsai, G., "Design Tool Integration: An Exercise in Semantic Interoperability", Proceedings of the IEEE Engineering of Computer Based Systems, Edinburgh, UK, March, 2000.

NASA, "CLIPS 6.0 Reference Manual", Software Technologies Branch, Lyndon B Space Center, Houston, Texas, 1992.

Pohl, J., "Information-Centric Decision-Support Systems: A Blueprint for Interoperability", Office of Naval Research (ONR) Workshop hosted by the CAD Research Center in Quantico, VA, June 5-7, 2001.

Pohl, J., A. Chapman, K. Pohl, J. Primrose and A. Wozniak, "Decision-Support Systems: Notions, Prototypes, and In-Use Applications", Technical Report, CADRU-11-97, CAD Research Center, Design Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA, January, 1997.

Pohl K., "Perspective Filters As A Means For Interoperability Among Information-Centric Decision-Support Systems", Office of Naval Research (ONR) Workshop hosted by the CAD Research Center in Quantico, VA, June 5-7, 2001.

Tolk A. and Muguira J. A., "The Levels of Conceptual Interoperability Model (LCIM)." Proceedings IEEE Fall Simulation Interoperability Work-shop, IEEE CS Press, 2003.

in expressive representation and collaborative architectures. Following an undergraduate degree in Computer Science, he earned Master's degrees in both Computer Science and Architecture. Over the past 20 years, he has provided technical leadership in the design and development of a number of context-oriented, decision-support systems for the US Department of Defense, including the Integrated Marine Multi-Agent Command and Control System (IMMACCS) for tactical command and control; the Joint Force Collaborative Toolkit (JFCT), supporting logistics planning for seabasing operations; and the TRANSWAY decision-support system, providing theater-to-theater logistics planning and visibility.

AUTHOR BIOGRAPHY



Kym J. Pohl is a senior software engineer and director of CDM Technologies, Inc. in San Luis Obispo, California. His current focus is agent-based, collaborative decision-support systems with particular interest