A Service-Sharing Methodology for Integrating COTS-Based Software Systems

Dean Jin
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada
djin@cs.umanitoba.ca

James R. Cordy
School of Computing
Queen's University
Kingston, Ontario, Canada
cordy@cs.queensu.ca

Abstract

This paper outlines a proposal for a novel approach to integration of COTS-based systems. Our methodology focuses on systems that operate in the same or similar application domains. Each COTS-based system is viewed in terms of the services it provides. We propose the use of a domain ontology and specially constructed service adapters to facilitate the sharing of these services with other COTS systems, effectively facilitating interoperability among them.

1. Introduction

The integration of existing software components into a consistent and interoperable whole is difficult to achieve in systems that were not originally designed to work together. This task can be even more challenging in systems composed of one or more COTS-based components. This is because internal system component integration adds a level of complexity to the external system integration challenges.

In this paper we outline an approach to systems integration that has demonstrated success in integrating tool systems that operate in the reverse engineering domain. It is our belief that this approach has promise for application in other domains, including higher-level integration as it relates to COTS-based systems. Our approach to integration relies on a number of fundamental assumptions about COTS systems and the motivation that drives the need for integration among them. We outline an operational and representational perspective on systems that leads to the development of an integration methodology that can be applied in COTS-based integration efforts.

To achieve effective interoperability, many organizational, technological and engineering issues must be fully addressed [7]. We restrict our discussion to information issues with special emphasis on sharing functionality as opposed to simply sharing data among systems. Technical issues will be the focus of our implementation efforts and future publications.

2. Terminology

The terminology used in the software development industry is notoriously inconsistent [3] and the area of COTS systems is certainly no exception. In this section we outline the terminology we use to characterize our approach to integration. This is done to ensure we use a consistent nomenclature, although this may not correspond to the broad range of terminology used in practice.

2.1. COTS Application Domains

COTS-based systems have been deployed extensively in business, government and military organizations. For this reason, COTS systems operate in a very broad range of application areas. However, integration efforts are typically driven by the need to combine functionality among systems that operate in the same or very similar application areas.

A COTS system typically falls into a set of one or more very similar application categories. We use the term *application domain* to refer to this set. Considering the application domain for a system makes it easier to determine the difficulty involved in integrating

two systems. Systems that operate in the same or very similar application domains typically represent similar concepts and often operate in a manner that is more like other systems that share the same application domain. Systems that operate in different application domains are typically more incompatible in what they represent and how they function.

At a conceptual level, there is typically some degree of overlap that exists between the operational and representational characteristics of COTS systems being integrated. Without such an overlap, the justification for integrating the systems would not exist. For example, it makes very little sense to integrate an accounting system with a weapons control system. More likely, these systems would be integrated with other business management and battlefield management systems respectively. Business management applications are typically transaction-based systems that work with financial or business rule concepts. In contrast, a system working in the battlefield management application domain might deal with concepts that relate to geographic position, troop or equipment logistics, and risk. It is clear that from a conceptual perspective an accounting system can be more easily integrated with other systems that operate in the business management application domain. This does not mean that integration cannot be accomplished among systems that operate in different application domains. It just means that the opportunity for equivalence among the concepts represented by each system is higher for those systems operating in the same application domain than it is for those operating in different domains.

2.2. Conceptual Equivalence

The notion of *conceptual equivalence* is important for our integration methodology. While many COTS systems function differently, when they operate in the same or very similar application domains they often represent the same concepts. Even though the underlying implementation for these representations may differ, from a conceptual perspective they represent the same thing. Conceptual equivalence among the representations of different COTS systems provides a means for achieving semantic interoperability, an important prerequisite for integration success.

2.3. Representation Knowledge and Invocation Knowledge

A COTS system can be viewed as a black box that provides certain functionality on its own or through its use with other previously deployed systems [1]. From this perspective, a COTS system can be characterized in terms of the inputs it requires and the outputs that it produces. One of the main barriers to COTS integration is that these systems really are black boxes. COTS vendors often have a vested interest in keeping the internal operations of their system proprietary. While stakeholders may not have access to the internal workings of a COTS system, they typically have knowledge of the input and output representations supported by the system. We call this *representation knowledge*. As well, stakeholders typically have access to the mechanisms that control the execution or invocation of a COTS system. We call this *invocation knowledge*. Representation and invocation knowledge provide a means for achieving functional interoperability, also an important prerequisite for integration success.

2.4. Services

The general notion of a *service* is defined by Lovelock *et al.* as:

> "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production" [6]

Using this definition, we can view the functionality provided by a given COTS-based system as a set of services. A COTS system may provide a single service or multiple services depending on how it is constructed. When given a valid set of required inputs, each service generates a corresponding output. In an integration context, this output is a desirable result that is relevant to the stakeholder of another system.

3. Abstracting the Functional Characteristics of a COTS System

If we take an abstract view of a COTS system, it can be divided into the following three components:

(a) **Schema**. A definition for the information model used by the system. Depending on the system, the information model for input may be different from the information model for the results provided by the system. For the purpose of our discussion, we distinguish the *input schema* from the *output schema* for a given COTS-based system. The *complete schema* is based on the union of both of these information models.

(b) **Operations**. A set of actions that the system performs based on the service invoked and the inputs provided. Different actions may be associated with different services.

(c) **Information**. This is data represented in a form dictated by the schema that is suitable for processing by the operations the system performs.

Of these three components, the schema for a COTS system is the most important. Since operations work on information input, their structure and form are inextricably tied to the representation defined by the schema. In this way, the schema acts as a regulator for both the information content and the data operations that the system performs. The schema also acts as a definitive reference for the representational and operational characteristics of a given system. It is therefore a key resource for integrators, providing representational and operational details about a COTS system.

The abstract functional characteristics of a COTS system can be expressed in terms of schemata, operations, and information as shown in Figure 1. The grey box is the COTS system itself. Although the inner workings of the system are unknown, we do know that a set of operations (Q) are built into the system. When a service is invoked by the user, a subset of these operations will be carried out on the input information (I_{in}) resulting in the output information (I_{out}). The input schema (S_{in}) defines the content and structure of the input information and the operations that are performed on them. The content and structure of the results (I_{out}) are defined by the output schema (S_{out}). A directed, dashed line reflects the important role the schema plays in defining the informational and operational aspects of the system.

4. Integration Architecture

The goal of our integration methodology is to foster interoperability among COTS systems through sharing of the services that each system provides. This can be accomplished when the requirements for semantic and functional integration have been fulfilled, namely:

- there is conceptual equivalence in the representation supported by each COTS system to be integrated,
- representation knowledge for each COTS system exists, and
- stakeholders have invocation knowledge for the services to be shared by the COTS systems.

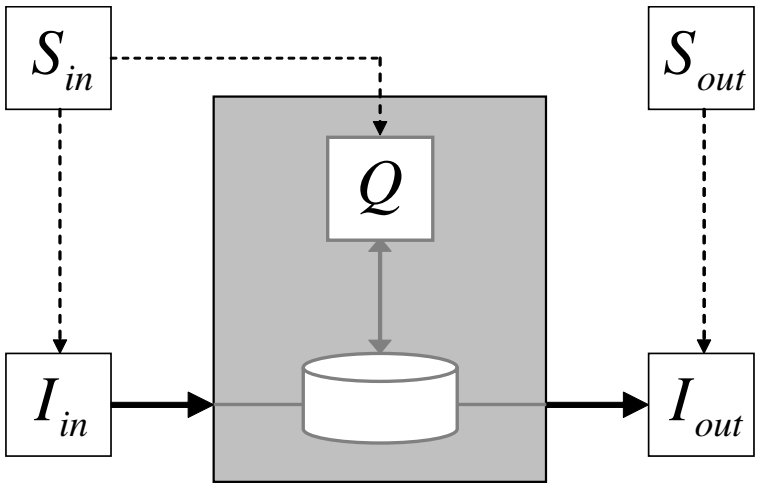
Consider n COTS-based systems that fulfill these requirements. We want these systems to cooperate in an *integration*, referring to the set of systems for which we are facilitating interoperability. A system in the integration is referred to as a *participant*. Each participant offers at least one service to the integration that can be shared among the other participants.

Figure 2 provides an architectural view of our proposed integration environment. The makeup of each system (shown as System 1, System 2, ... System n) participating in the integration reflects our view of the abstract functional characteristics of COTS systems we outlined in Figure 1. Each system has its own input and output schema (S_{in} and S_{out}) that defines the content and structure of the input and output information (I_{in} and I_{out}).

Our integration methodology involves the creation of two types of components:

1. **Domain Ontology**. This component stores all the knowledge required to support service-sharing among each of the systems participating in the integration. The knowledge is stored as a tabularized, cross-referenced compilation of representational concepts and services offered by each integration participant.

   Taken together, the representational concepts stored in the domain ontology define a conceptual space, consisting of conceptual slots that data fit into. A data item fits into a slot only when the



S_{in} = Input schema
S_{out} = Output schema
Q = A set of operations the COTS system implements
I_{in} = Information input into the COTS system
I_{out} = Resulting information output from the COTS system

Figure 1. Abstract Functional Characteristics of a COTS System

Each COTS system can be viewed as a black box. Although the implementation details of the system are hidden, some knowledge (either specific or intuitive in nature) of the set of operations (Q) that the system carries out is available. The structure and format of the input information (I_{in}) and the resulting output (I_{out}) are defined respectively by schemas S_{in} and S_{out}. The operations that the system performs on the input information depends on the characteristics of the information model defined by the input schema.

---

concept it represents matches a concept in the domain ontology. We say that a system has *concept support* when this occurs. We describe concept support (in the context of reverse engineering tool integration) in more detail in an earlier paper [5]. Shared services only operate on data items that fit into these conceptual slots.

A service offered by a system participating in the integration can be shared only when the concepts required by the service intersect with the concepts supported by another participant. Only one domain ontology is required for each integration implementation.

2. **Conceptual Service Adapters (CSA$_n$)**. These components function as integration facilitators for systems participating in the integration. Each COTS system is affiliated with a single conceptual service adapter. Each makes use of the domain ontology to get the information it needs to regulate the integration process. Conceptual service adapters perform the following three main functions:

   (a) *Shared Service and Concept Support Identification*. Making use of the knowledge stored in the domain ontology, each conceptual service adapter identifies requests for shared services and determines the concepts each service requires.

   (b) *Data Filtering*. Depending on the mode of operation invoked, each conceptual service adapter will map all data items into and out of the conceptual space defined by the domain ontology. We call this process filtering. Mapping data into the conceptual space

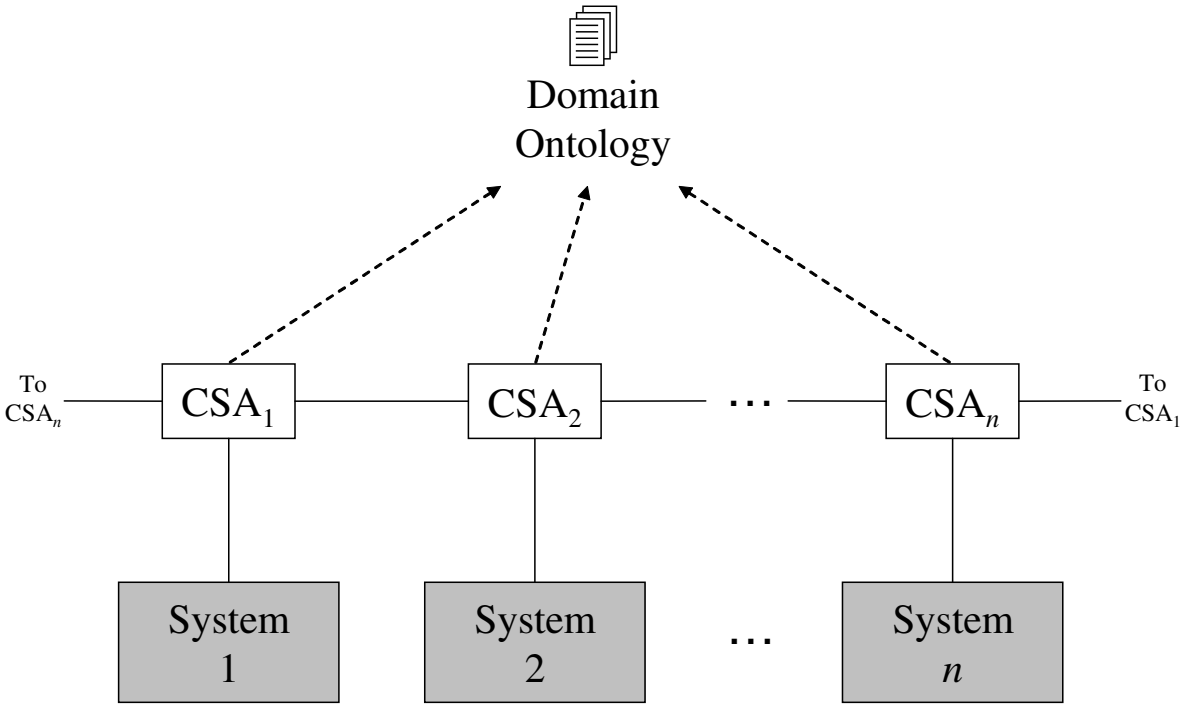


Figure 2. Our COTS System Integration Architecture

Each system participating in the integration supports the representation of concepts that relate to the application domain they operate in. The Domain Ontology stores knowledge about the concepts each system supports, the shared services each system offers to the integration, and the concept support that each service requires. On request, a Conceptual Service Adapter (CSA) uses the knowledge stored in the Domain Ontology to identify a service that can be shared. Each CSA looks after the data filtering required for a given system and communicates with the CSA for the other system to request execution of the shared service.

---

is performed by an inFilter. Mapping from the conceptual space is performed by an outFilter. Both of these filters are specially tailored to work with the representation supported by the information model for the system that the conceptual service adapter is associated with. A more detailed discussion of the filtering process (in the context of reverse engineering tool integration) can be found in our earlier paper [4].

(c) *Shared Service Execution*. Each conceptual service adapter manages requests from other conceptual service adapters for the execution of shared services on the system they are associated with.

Although all the conceptual service adapters have the same basic architecture and operating characteristics, each is specially constructed to handle the functional and information filtering aspects of its corresponding system that are required to facilitate interoperability.

The access and communication links between the domain ontology, the conceptual service adapters, and systems they are associated with are shown as undirected, solid black lines in Figure 2.

5. An Integration Example

To better understand how our approach to COTS system integration works, we provide an example that

applies our methodology to a COTS system integration problem.

Consider a (hypothetical) organization that provides relief supplies to remote areas of a war-torn country. The organization has a number of trucks for transporting equipment and supplies to areas where they are needed. Each truck driver takes a predefined route from the supplies depot to the destination. A COTS-based Relief Logistics System supports distribution and route planning activities, keeping track of the trucks, their drivers, the supplies they carry and the routes they take.

Although the inner workings of the Relief Logistics System are hidden, the relief organization does have access to the file where all the information maintained by the system is stored. An analysis of this file yields the schema shown in Figure 3(a). The representation is expressed as an *Entity-Relationship (E-R)* [2] model showing the type of entities and relationships that exist among the information supported by the system. The Relief Logistics System is executed from a command shell by calling a system function and supplying a set of appropriate parameters. These commands, essentially the services offered by the Relief Logistics System, are shown in Figure 3(b).

Transporting relief supplies can be dangerous, especially in a war-zone where the position of conflict areas changes by the minute. Although the Relief Logistics System is effective for planning the route for a supply truck, it does not work in real-time. At the start of each day the system assigns a driver and supplies to a truck. After the truck is loaded, the system prints the directions for the driver based on the route calculated and the delivery commences. During the day, changes in the conflict situation often result in deliveries entering danger zones. To prevent these situations, the relief organization has been searching for a system that will allow them to detect dangerous routes before a delivery commences and respond to immediate threats as they come about during the day.

Another (hypothetical) relief organization working in the region has a COTS-based Threat Management System that they use to assist in their operations. This real-time system is constantly updated based on reports from field observations. They are willing to allow the Relief Logistics System to work with the Threat Management System. Both organizations possess the hardware infrastructure to support communication between each of the systems. The only barrier to the arrangement is the software interoperability aspect of the integration.

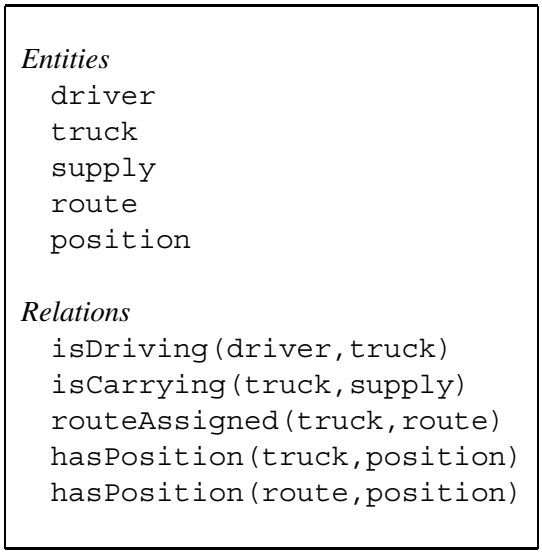
Unlike the Relief Logistics System, the Threat Management System has two modes of operation. An operator console runs all the time, providing the user with an interactive environment where a database of threat positions are recorded and updated. The system also has an on-demand database interrogation operation that accepts a stream of position coordinates and produces a report based on any threat identified. It also returns a stream of 2-tuples representing threatened positions and an indicator of the type of threat that exists at that position. Externally, the COTS-based system only has a representation for positions and threat status indicators. The schema for the Threat Management System is shown in Figure 4(a). While the internal operations of the threat management console are inaccessible, access is provided to the `checkThreat` service, whose calling parameter is shown in Figure 4(b) as a stream (indicated by a `*`) of `position` values.

5.1. Consideration of Requirements

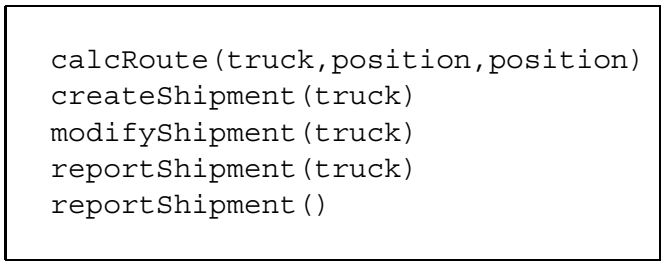
Our integration methodology can be used to facilitate interoperability between the Relief Logistics System and the Threat Management System. In particular, we want to share the `checkThreat` service offered by the Threat Management System by applying it to information that comes from the Relief Logistics System. We see from the problem statement that the requirements for semantic and functional integration for both systems have been met. While each system might represent positions differently (for example, one might use a degree decimal format while the other uses a Universal Transverse Mercator format), they both represent the *position* concept in a semantically equivalent manner. Schemas for both systems provide the representation knowledge required. Both organizations have invocation knowledge for the service to be shared by the systems.

5.2. Creating the Domain Ontology

The first step in implementing the integration involves the creation of the domain ontology. This in-

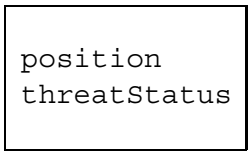


(a) Relief Logistics System Schema

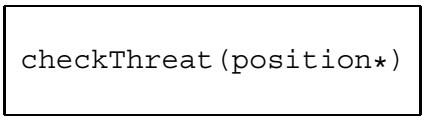


(b) Relief Logistics System Services

Figure 3. Representational and Service Characteristics of the Relief Logistics System



(a) Entities Represented



(b) Service Offered

Figure 4. Representational and Service Characteristics of the Threat Management System

volves identifying and organizing all the representational and service related concepts supported by each of the systems participating in the integration. In our experience integrating reverse engineering tools, the creation of the domain ontology is the most difficult and time consuming step in the integration process. Size and representational diversity in the schemas make this a difficult task. A simple domain ontology for our integration example is shown in Figure 5. Figure 5(a) informally defines all the representational concepts supported by all participants in the integration. For simplicity, we represent these concepts as an E-R model representing conceptual entities and relationships. The *Services Dictionary* shown in Figure 5(b) outlines the relationship between each service, the system that offers it and the concepts it requires. The *Systems Dictionary* shown in Figure 5(c) indicates the relationship between each system and the concepts it supports. After the domain ontology is created on paper it is codified and stored in a machine readable form.

5.3. CSA Construction

The next step involves the construction of a conceptual service adapter for each system participating in the integration. Each adapter manages all aspects of the integration as it relates to the system it corresponds to. The construction of inFilter and outFilter components is an important part of this process. The inFilters

Entity Concepts

person
A uniquely identifiable human individual

position
The coordinates of a fixed point.

status
An indicator for a condition that exists.

supply
Material goods that provide relief to needy recipients.

vehicle
A uniquely identifiable transportation machine.

Relational Concepts

Assignment
An entity is entrusted to look after another entity.

Containment
An entity holds another entity within it.

Haul/Carry
An entity loads and transports some other entity.

Use
An entity utilizes or occupies another entity.

(a) Representational Concepts

Service	Offered By	Requires Concept
CalculateRoute	Relief Logistics System	position (x2) vehicle
checkThreat	Threat Management System	position

(b) Services Dictionary

COTS System	Supports Concept
Relief Logistics System	person position supply vehicle Assignment Haul/Carry Use
Threat Management System	position status Containment

(c) Systems Dictionary

Figure 5. Domain Ontology

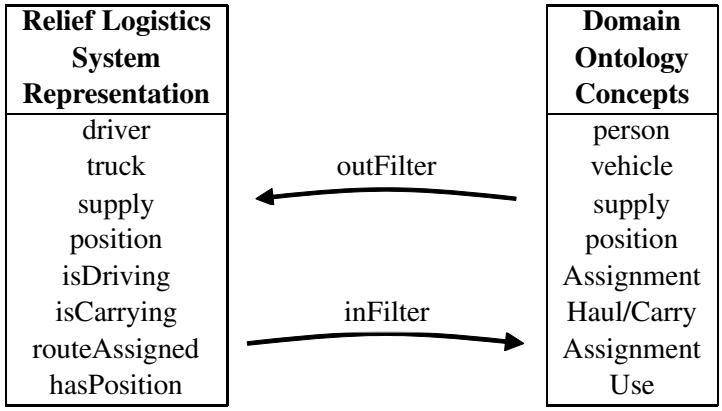


Figure 6. Relief Logistics System - Conceptual Service Adapter Filter Mappings

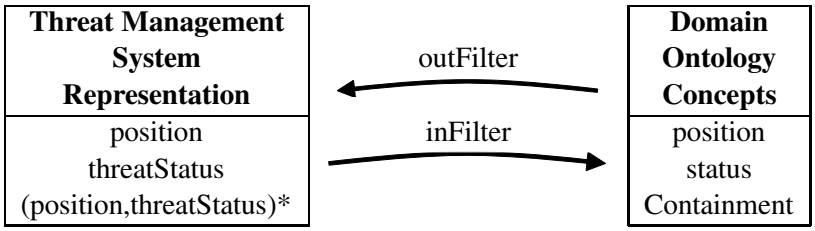


Figure 7. Threat Management System - Conceptual Service Adapter Filter Mappings

manage the mapping of information from each system to concept slots in the conceptual space defined by the domain ontology. The outFilters do the same mapping in the opposite direction. The inFilter and outFilter mappings performed by the conceptual service adapters for the Relief Logistics System and the Threat Management System are shown in Figures 6 and 7 respectively.

5.4. How the Integration Works

Once constructed, the domain ontology and conceptual service adapters for each system handle all the functionality required to facilitate interoperability between each system. The following scenario outlines how this is achieved:

- The Relief Logistics System is used to assign a driver and supplies to a truck. A route for the shipment is determined using the calcRoute function native to the system. The next step involves determining if the route is safe. The user requests the `checkThreat` service from the conceptual service adapter (CSA) for the Relief Logistics System. The CSA looks up the `checkThreat` service in the domain ontology and identifies the Threat Management System as the service provider. It also determines that the `checkThreat` service requires support the position concept. The CSA then verifies that the Relief Logistics System supports the position concept.
- The CSA for the Relief Logistics System uses its inFilter to map the information stored in the system to the conceptual space. For example, all driver entities are mapped to the person concept, all positions entities are mapped to the position concept, and all hasPosition relationships are mapped to the Use concept. After the mapping is complete, the Relief Logistics System CSA sends a message to the Threat Management System CSA requesting the `checkThreat` service.
- The Threat Management System CSA receives the request and uses its outFilter to map the information from the conceptual space that the

`checkThreat` system requires. In this simple integration example, only the position concepts are relevant. After the mapping is complete, the CSA organizes the position information into a stream and supplies it to the `checkThreat` operation.

- The `checkThreat` operation generates two sets of results. The first is a report that indicates all the positions on the route where is a threat is present. This is the result the Relief Logistics System user is looking for. The second result is a stream of 2-tuples representing threatened positions and an indicator of the type of threat that exists at the position. This information is collected by the Threat Management System CSA, which uses its inFilter to map the data to the conceptual space. Each postion-threatStatus pair is mapped to the Containment relationship concept in the conceptual space. The CSA then sends a message to the Relief Logistics System CSA that the `checkThreat` service has completed and the integration ends.

It is notable that no native representation for the results returned by the `checkThreat` service is supported in the Relief Logistics System. Although conceptual equivalence exists between both systems in relation to the position concept, this only permits interoperation in one direction. Fully bi-directional interoperability would require a modification to the Relief Logistics System to support a representation for the threat information provided by the Threat Management System.

We also note that semantic equivalence is a difficult problem in system integration. Our example demonstrates service-sharing based on a very simple concept (position). In practice [4] we have found that very subtle differences in the way services are implemented can yield counter intuitive results, even when dealing with concepts that are semantically equivalent. In future work we will be exploring the deeper role of semantics in the context of our approach to integration.

6. Conclusion

In this paper we have outlined a proposal for a service-sharing approach to integration for COTS software systems. In practice, we have used these methods to successfully integrate reverse engineering tool systems. The limited example shown demonstrates that our approach holds promise in more general integration contexts as well. We look forward to applying our methodology to real-world COTS system integration problems to further explore the advantages and limitations of our integration methodology.

References

[1] B. Boehm and C. Abts. "COTS Integration: Plug and Pray?". *IEEE Computer*, 32(1):135–138, January 1999.

[2] P. P.-S. Chen. "The Entity Relationship Model Toward a Unified View of Data". *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[3] A. M. Davis. *Software Requirements: Objects, Functions and States*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[4] D. Jin and J. R. Cordy. "Factbase Filtering Issues in an Ontology-Based Reverse Engineering Tool Integration System". In *Proceedings of the 2nd International Workshop on Metamodels, Schemas and Grammars for Reverse Engineering*, volume 137 of *Electronic Notes in Theoretical Computer Science*, pages 65–75, Delft, Netherlands, October 2004.

[5] D. Jin, J. R. Cordy, and T. R. Dean. "Transparent Reverse Engineering Tool Integration Using a Conceptual Transaction Adapter". In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR 2003)*, pages 399–408, Benevento, Italy, March 2003.

[6] C. Lovelock, B. Lewis, and S. Vandermerwe. *Services Marketing: A European Perspective*. Prentice-Hall, London, 1996.

[7] D. Smith. Guide to Interoperability. Integration of Software Intensive Systems (ISIS) Initiative, Software Engineering Institute, 2005. URL: `http://www.sei.cmu.edu/isis/guide/isis-guide.htm`.